AFRL-IF-RS-TR-2002-162
Final Technical Report
July 2002

JFACC – AC2C EXPERIMENT PLAN AND DEFINITION

Applied Research Laboratory

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J111

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JULY 2002	**3. REPORT TYPE AND DATES COVERED** **Final** **Aug 99 – Feb 01**

4. TITLE AND SUBTITLE JFACC AC2C EXPERIMENT PLAN AND DEFINITION	**5. FUNDING NUMBERS** C - F30602-99-1-0547 PE - 63760E PR - J111 TA - 00 WU - 01
6. AUTHOR(S) Shashi Phoha	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Applied Research Laboratory Penn State University, P.O. Box 30 State College PA 16804	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714 AFRL/IFSA 525 Brooks Road Rome NY 13440-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2002-162

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Carl DeFranco/IFSA/(315) 330-3096

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

This research documents a theory that models Command and Control (C2) as control systems.

This work designed, implemented and tested algorithms in the continuous domain to search for and attack targets, and to find and escort damaged friendly aircraft back to base.

The research resulted in tools for building and testing controllers, and a test-bed for measuring the performance of control systems performing C2 under simulated battlefield conditions. Experiments were planned around the design and evaluation of discrete event control hierarchies of one, two and three layers. Within each set, experiments measured the performance as a function of the experimental variables in the battlefield simulator.

The results provide evidence that C2 policies can be implemented as control systems and that hierarchical control systems can shield commanders from information overload.

14. SUBJECT TERMS JFACC, controller validation, command and control, simulated battlefield experiments	**15. NUMBER OF PAGES** 70
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. BACKGROUND 1

1.1. Controller Validation Experiments 2

1.2. Performance Evaluation Experiments 2

1.3. Red Blue Experiments 3

2. STATEMENT OF EXPERIMENTAL OBJECTIVES 3

2.1. Controller Validation Experiments 3

2.2. Performance Evaluation Experiments 8

2.3. Red Blue Experiments 11

2.4. Value of Experiments to the DoD 12

3. DESCRIPTION OF EXPERIMENTAL SETUP 13

3.1. Simulation Features 14

3.2. Variables or Correlated Parameters 16

3.3. Specification of Test Runs 16

4. PRE-LAB ANALYSIS 21

4.1. Predictions 21

4.2. Validation Criteria 21

4.3. Evaluation Criteria 21

5. RESULTS & DISCUSSION 23

5.1. Controller Validation Results 23

5.2. Performance Results 30

5.3. Red-Blue Experimental Results 36

5.4. Theoretical Results 38

5.5. Algorithmic Results 41

5.6. Other Experiments 44

APPENDIX A: OBJECTIVE FUNCTION 49

APPENDIX B: MISSION OVERVIEW **51**

APPENDIX C: CURRENT CONTROLLER SPECIFICATIONS **54**

Experiment C1 – Controller for a single wild weasel **54**

Experiment C2--Fixed Two-Level Controller Hierarchy for Wild Weasels and **57**

Experiment C3 – Dynamic Three-Level Controller Hierarchy for Wild Weasels **61**

Table of Figures

FIGURE 2.1: HIERARCHICAL CONTROL 5
FIGURE 2.2: HIERARCHICAL CONTROL OF MULTIPLE LOW-LEVEL CONTROLLERS 6
FIGURE 2.3: CONTROL HIERARCHY SCALABILITY TEST 7
FIGURE 2.4: ABSTRACTED LOW LEVEL CONTROLLERS FOR STATIC AND DYNAMIC HIERARCHIES 8
FIGURE 3.1: CONCEPTUAL VIEW OF CYBERLAND SCENARIO 14
FIGURE 3.2: HIGH-LEVEL FLOW OF EXPERIMENT EXECUTION. 18
FIGURE 3.3: EXPERIMENT EXECUTION FLOW 19
FIGURE 5.1: CONTROLLABILITY OF PLANT MODEL POPULATIONS 24
FIGURE 5.2: ROBUSTNESS FOR SINGLE LEVEL CONTROLLER 24
FIGURE 5.3: RECOGNITION OF ACTUAL PLANT LANGUAGE BY PLANT MODEL 25
FIGURE 5.4: CONTROLLABILITY OF EACH HIERARCHICAL LEVEL OVER PLANT POPULATION BY GENERATION 26
FIGURE 5.5: ROBUSTNESS FOR LOWER LEVEL CONTROLLER 26
FIGURE 5.6: ROBUSTNESS FOR SECOND LEVEL CONTROLLER 27
FIGURE 5.7: RECOGNITION OF ACTUAL PLANT LANGUAGE BY PLANT MODEL FOR A FIXED HIERARCHY 27
FIGURE 5.8: ROBUSTNESS FOR THIRD LEVEL CONTROLLER 28
FIGURE 5.9: RECOGNITION OF ACTUAL PLANT LANGUAGE BY PLANT MODEL FOR A FIXED HIERARCHY 29
FIGURE 5.10: EVENTS PER RUN FOR EACH LEVEL OF THE HIERARCHY 29
FIGURE 5.11: ROBUSTNESS AND PERMISSIVENESS FOR EACH LEVEL OF THE HIERARCHY 30
TABLE 5.1: EXPERIMENT P1 RESULTS 30
TABLE 5.12: EXPERIMENTAL P1 RESULTS 31
TABLE 5.2: EXPERIMENT P2 RESULTS 31
FIGURE 5.13: EXPERIMENT P2 RESULTS 32
TABLE 5.3: EXPERIMENT P3 RESULTS 32
FIGURE 5.14: EXPERIMENT P3 RESULTS 33
TABLE 5.4: EXPERIMENT P4 SINGLE LEVEL CONTROLLER RESULTS 33
TABLE 5.5: EXPERIMENTAL P4 TWO LEVEL SEMI-STATIC CONTROLLER 34
FIGURE 5.15: PROPORTION OF CORRIDOR CLEARED VS. TARGET KILL PROBABILITY 34
FIGURE 5.16: DYNAMIC HIERARCHY VS. SINGLE LEVEL CONTROLLER 35
TABLE 5.6: EXPERIMENT P5 –THREE LEVEL HIERARCHY 35
FIGURE 5.17: THREE LEVEL DYNAMIC HIERARCHY 36
FIGURE 5.18: EXPERIMENT RB1 RESULTS 37
FIGURE 5.19: EXPERIMENT RB1 RESULTS 37
FIGURE 5.19: EXPERIMENT RB4 RESULTS 38
FIGURE 5.20: REPRESENTATION OF A FORMAL LANGUAGE WITH ALPHABET A,B 40
FIGURE 5.21: ENTROPY OF THE MOST FREQUENT SUBSTRINGS IN A SIMULATION 41
TABLE 5.6: AVERAGE RESULTS OF EXPERIMENTS RUN WITH THIRTY ENEMY TARGETS. 42
FIGURE 5.22: HIGH RISK SITUATION. 43
FIGURE 5.23: WILD WEASEL PLATFORM ROUTE CHOSEN TO MINIMIZE RISK. 43
FIGURE 5.24: LANGUAGE DOMAINS FOR SIMILAR PLANT MODELS 44
FIGURE 5.25: NUMBER OF SIMILAR PLANTS PER GENERATION. 45
FIGURE 5.26: THE CONTROLLABILITY OF EACH GENERATION IN $\mathcal{G}$ 45
FIGURE 5.27: RESULTS FROM TWO AIRCRAFT ONE-TARGET SCENARIOS. 46
FIGURE 5.28: RESULTS OF INITIAL EXPERIMENTS 47
FIGURE 5.29: SENSITIVITY OF WILD WEASEL PERFORMANCE TO ORDANCE RANGE 47
FIGURE 5.30: EARLY LARGE SCALE SEAD EXPERIMENTS 48
FIGURE 5.31: CURRENT VISUALIZATION INTERFACE 48
FIGURE C.1: TRANSITION DIAGRAM OF FSM FOR WILD WEASEL PLANT MODEL 54
TABLE C-1. STATES AND EVENTS FOR WILD WEASEL PLANT MODEL 55
TABLE C.2: CONTROLLER REQUIREMENTS FOR A SINGLE *WILD WEASEL* 55

FIGURE C.2: CONTROLLER FOR A SINGLE *WILD WEASEL* 56
FIGURE C.3: STEPS FOR TESTING CONTROLLABILITY 56
FIGURE C.4: LOWER LEVEL PLANT MODEL FOR HIERARCHICAL CONTROL 57
TABLE C.3: LOWER LEVEL EVENTS AND STATES FOR HIERARCHICAL CONTROL 58
TABLE C.4: LOWER LEVEL CONTROL SPECIFICATIONS FOR HIERARCHICAL CONTROL 58
FIGURE C.5: PLANT MODEL FOR SECOND LEVEL OF HIERARCHICAL CONTROL 59
TABLE C.5: EVENTS, STATES, AND MAPPINGS FOR LEVELS 1 AND 2 OF HIERARCHICAL CONTROL 59
TABLE C.6: SECOND LEVEL CONTROL SPECIFICATIONS (FOR THE OPERATION OF A GROUP OF AIRCRAFT) 60
FIGURE C.6: SECOND LEVEL CONTROLLER 60
FIGURE C7: TOP LEVEL PLANT FOR 3 LEVEL HIERARCHY 61
TABLE C.7: STATES AND EVENTS FOR TOP LEVEL CONTROLLER 61
TABLE C.8: MEALY MACHINE CONSTRUCTION FOR MIDDLE-LEVEL INDIVIDUAL GROUP OF TWO AIRCRAFT 62
TABLE C.9: THIRD LEVEL SPECIFICATIONS 62
FIGURE C.8: THIRD LEVEL CONTROLLER DIAGRAM 62

Executive Summary:

This document presents the accumulated research results of the ARL JFACC Program. We have developed a theory that models Command and Control (C2) as control systems.

We have designed, implemented and tested algorithms in the continuous domain to search for and attack targets, and to find and escort damaged friendly aircraft back to base. The domain of discrete event control is relatively new, and we have achieved breakthroughs in three areas. The first is in measuring the formal languages of discrete variables. The second is in measuring the performance of the controller in terms of Robustness and Permissiveness and the third is in the abstraction of formal languages using modified Shannon entropy.

Our research has resulted in tools for building and testing controllers, and a test-bed for measuring the performance of control systems performing C2 under simulated battlefield conditions. The experiments were planned around the design and evaluation of discrete event control hierarchies of one, two and three layers. Within each set, experiments measured the performance as a function of the experimental variables in the battlefield simulator.

The results provide evidence that C2 policies can be implemented as control systems and that hierarchical control systems can shield commanders from information overload.

JFACC - AC^{2}C Experiment Plan and Results

1. Background

The goal of the DARPA JFACC program is to extend control theory, and the use of the feedback control loop, into all levels of air campaign planning and execution. This will change current C^2 practice by merging planning and execution into one integrated system.

The Applied Research Laboratory of the Pennsylvania State University has developed a framework of interacting C^2 agents and their associated intelligent controllers to implement this concept. The physical battle-space, or a simulation thereof, forms the plant on which the controllers act.

The air campaign planning and execution process has been modeled as a hierarchical network of finite state automata (FSAs). Modularity allows the hierarchy to be changed dynamically to express reconfiguration of a proposed campaign. Transformations on the FSA network dynamically produce C^2 hierarchies as the campaign evolves. Intelligent discrete event controllers use the FSAs to suggest appropriate actions for friendly campaign participants.

Adaptive Command and Control Coalitions (AC^{2}Cs) are consistent with the JFACC goal to catalyze a revolutionary change in military C^2 through the agile and stable control of distributed and dynamic military operations conducted in an uncertain and rapidly changing environment. Our approach establishes an interacting hierarchy of controllers and "tactical" intelligence algorithms that integrates planning and execution of air campaigns. It supports the JFACC goal in the following ways:

- The stability of each controller can be shown analytically.
- A relatively high degree of local autonomy allows a quick response to any event.
- Feedback is present at all levels of campaign C^2 to coordinate individual platforms and ensure the achievement of overall mission goals.
- The use of adaptive coalitions allows the C^2 to adapt to changes in the environment.

Air campaign planning is a challenging problem, based on coordinating multiple entities with complex interactions and dependencies. This requires experiments at multiple levels of control. Experiments and testing require running war game simulations at all levels of mission planning and execution.

The experiments will be based on a *corridor-clearing scenario.* This is a simple scenario. Assault corridors are defined containing enemy SAM's. SAM's are both mobile and fixed. Friendly forces will send a set of *Wild Weasel* aircraft to find and destroy the SAM's. All enemy air defenses within the corridor are to be disabled for the length of the mission.
We will investigate the effects of variations and uncertainty in the plant model with regard to a number of experimental variables including:

- The degree of uncertainty in sensing the environment: This includes missed and false alarms, and uncertainty in the location, number, and characteristics of friendly and enemy platforms. It will result in the detection of hostile forces while the missions are underway. The Discrete Event Controllers need to handle reaction to unforeseen events. This experiment will explore reaction dynamics of the friendly forces as the mission deviates from presupposed mission specifications.
- The degree of uncertainty in communications: This includes missed or erroneous communications between all levels of the C2 hierarchy. Decisions are made to allow the system to recover from these communications faults. Decisions must be made promptly and the time is not available for finding strictly optimal decisions. This research considers how to make good decisions within strict time limits.
- Parameters in the mission and tactical intelligence: Mission parameters include the scale of the mission, i.e., the number of aircraft both sides and the effect of small variations in the initial conditions of the simulation. Tactical intelligence parameters include the type of optimization used for finding targets, routing planes, allocating resources, *etc.*

This document describes the experimental plan, which will provide a description of our research.

1.1. Controller Validation Experiments

- **Experiment C1** ***Basic control of individual platforms without the use of a hierarchy.* –** This is an experiment to confirm the research done on this project on the design of individual controllers. Simulations will be done using one or more controllers for individual platforms that are not coordinated through a command and control hierarchy. The results will be analyzed to determine goodness of fit measurements for control systems such as controllability, permissiveness, and robustness. It will include methods based on research done on this project to extend measurements defined on continuous control systems to discrete control systems.
- **Experiment C2** ***Hierarchical control of individual platforms.* –** This experiment will confirm the research done on fixed hierarchies of controllers. It will test whether the control hierarchies satisfy the basic controller requirements described in **C1.** In addition, the experiment will determine whether the control hierarchies are scalable and consistent.
- **Experiment C3** ***Adaptive Coalitions.*** – This is experiment will confirm the research done on the design of dynamic hierarchies of controllers. These systems can change the structure of the hierarchy during the mission in response to events in the simulation. In addition to testing the basic controller requirements in **C1** and the hierarchical control requirements in **C2**, the experiment will determine whether the use of a dynamic hierarchy improves the performance of the blue team.

1.2. Performance Evaluation Experiments

- **Experiment P1** ***The Effect of Sensor Quality Degradation in Wild Weasel Planes on Corridor Clearing Performance*** – We vary the sensor quality of the *Wild Weasels* within the *corridor clearing scenario* defined above.
- **Experiment P2** ***The Effect of Sensor Quality Degradation in Supervisory Planes on Wild-Weasel Corridor Clearing Performance*** – Within the previous scenario, the sensor quality of the supervisory aircraft is varied. The affect this has on the system's ability to complete its mission is measured.

- **Experiment P3** ***The Effect of Damage Assessment Degradation on Wild-Weasel Corridor Clearing Performance*** – In the same scenario as P1 and P2, the ability of the *Wild Weasel* to detect damage to the target is varied.
- **Experiment P4** ***The Effect of SAM Lethality on Wild Weasel Corridor Clearing Performance in One and Two Level C2 Hierarchies*** – *Wild Weasel* aircraft will be tasked with clearing two assault corridors as in P1 and P2. Target lethality will be altered and the effect on system performance measured in both one and two level hierarchies.
- **Experiment P5 The** ***Effect of Multi-level, Dynamic Hierarchies on Corridor Clearing Performance***—*Wild Weasel* aircraft will be tasked with clearing a single assault corridor. Target lethality will be modified and performance assessed. Aircraft will be organized into a three level, dynamic hierarchy. These results will be compared to the results of Experiment P4.

1.3. Red Blue Experiments

- **Experiment RB1** ***Performance evaluation of blue forces (Wild Weasels) in response to red forces (SAMs) using different strategies. -*** Controllers will be implemented for the red forces (SAMs). Different strategies are being implemented for red. Monte Carlo simulations will be used to analyze the effects of these strategies on the blue controller.

2. Statement of Experimental Objectives

2.1. Controller Validation Experiments

Hypothesis C1:
The controllers developed in this project are relatively robust and capable of controlling the nominal plant model.

Level of C1:
- **Level 1** – Design a *nominal plant model* of a *Wild* Weasel aircraft implemented as an FSA. Design a controller, also based on an FSA so that the controller operates the nominal plant model within a set of specifications without deadlocks or livelocks.
- **Level 2** – Verify the controller. Define a measure function on a space of formal languages and use it to estimate the permissiveness and robustness of the controller.
- **Level 3** –Operate the controller. Run a set of simulations with the controller and the *actual plant*, i.e. ARL's battlefield simulator. Verify that the controller operates the actual plant within a set of specifications without significant deadlocks or livelocks.

Description C1:
In DES control, the plant is as a state machine that processes symbols from an alphabet in a way that forms a formal language, L. The alphabet, Σ, can be partitioned into symbols corresponding to uncontrollable events, Σ_u, and symbols corresponding to controllable events, Σ_c. An example of a controllable event is a friendly aircraft firing at an enemy target, while an example of an uncontrollable event is the enemy target firing on the friendly aircraft.

The controller can be thought of as a recognizer of uncontrollable events and a generator of controllable ones. In analogy to continuous control systems, the controllable events are the plant input (feedback), and the uncontrollable events are the plant output.

The goals of the control system are given as a set of specifications such as, "*If the aircraft runs out of weapons, it returns to base*." The specifications can be formalized as a language *K*, which is a subset of the language *L*. The controller attempts to enforce the specifications by restricting the plant to operate within *K*, rather than *L*. It can only enable or disable controllable events. The plant is said to be controllable with the controller if it can operate the plant in a way that satisfies the specifications without going into a deadlock or livelock.

The controllability definition can be formalized as: K controls a plant with language L if $\overline{K}\Sigma_u \mathrm{I} \; L \subseteq \overline{K}$, where $\overline{K}$ is the prefix closure language of *K* [Ramadge & Wonham 91]. That is, the occurrence of any uncontrollable event at any time permitted by the dynamics of the plant, will not violate the control specifications.

In our research however, the actual plant is never completely known. It is represented by an FSA, called the nominal plant model that contains all of the available knowledge about the actual plant. The language of the nominal plant model approximates the language of the true plant; the controller is designed on this basis.

It is therefore important to determine whether the controller can control languages other than the one upon which it was designed. This quantity is known as robustness. The more permissive the controller, the more likely it is to be able to control the actual plant.

We have determined a method to estimate permissiveness through the development of two concepts, a population of plant languages near the language of the nominal plant model, and a measure for formal languages of a given alphabet, Σ.

The population of plant languages can be defined as the languages of a set of plants, R, derived from the nominal plant model through the application of a small number of primitive graph operations such as the addition or deletion of a single state or transition, under the restriction that the resultant FSA is deterministic.

We define Σ^* as the set of all finite strings of characters from the alphabet Σ and construct a measure on the power set of Σ^* :

$$\mu(L) = \sum_{i=1}^{\infty} W_i N_i(L)$$

where L is a formal language, $\mu(L)$ is its measure, $N_i(L)$ is the number of strings of length i in language L, and W_i is a weighting factor for strings of length i.
The robustness of a controller, *C*, can then be defined in terms of the measure function. This will be discussed further in Section 5.3.

We cannot prove the controllability of the actual plant, because it is not precisely known. We will observe the results of simulations with the controller and actual plant to ensure that controllability is not violated during any of the runs.

Hypothesis C2:
The fixed hierarchies of interacting controllers developed in this project are hierarchically consistent, permissive, and scalable. They also reduce the *information load* at the upper levels of the hierarchy.

Level of C2:

- Level 1 – Design controllers for a fixed hierarchy that are *hierarchically consistent* (the hierarchical equivalent of *controllable*) with respect to the nominal plant model.
- Level 2 – Verify the controllers for a fixed hierarchy. Determine the permissiveness, robustness and hierarchical consistency of the hierarchy.

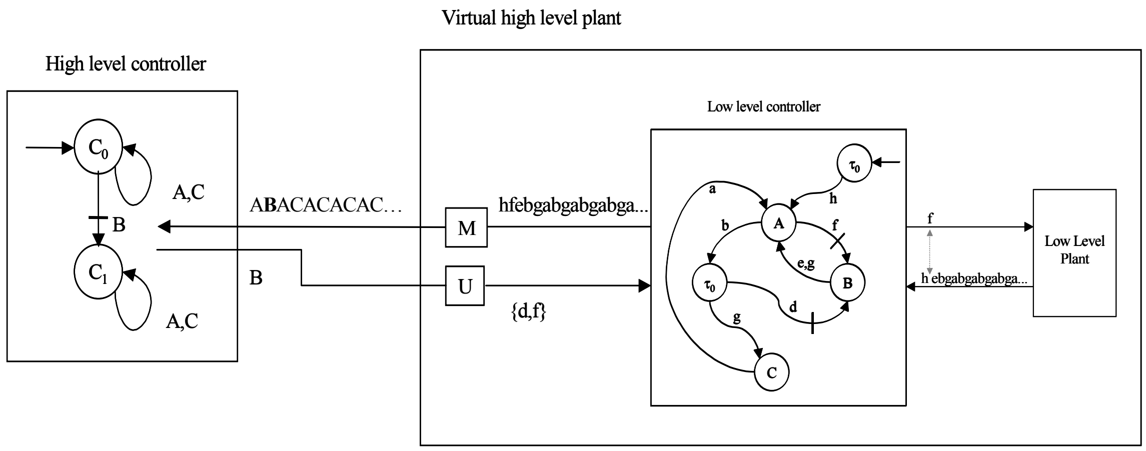


Figure 2.1: Hierarchical Control

- Level 3 – Operate the fixed control hierarchies. Run a set of simulations with the hierarchical controller and the *actual plant*, i.e. ARL's battlefield simulator. Verify that the controller operates the actual plant within a set of specifications without significant deadlocks or livelocks. Determine the "information load" at the supervisory nodes in the hierarchy. This will be estimated as the number of events per node per unit of simulated time.

Description C2:
Hierarchical structures are natural for C^2 applications and are being used to control dynamic systems for a variety of tasks. Control is divided between higher levels, which process events of greater generality and larger scope; and lower levels that process events of lesser scope.

This notion has been formalized in a way that is illustrated in Figure 2.1. The higher level controller is designed to control a virtual high level plant with the following components: the low-level, i.e. *actual*, plant; the low-level controller; *M*, a mapping from strings of the low-level plant language to strings of the high-level plant language; and *U*, a mapping from high-level, controllable events to low-level *control patterns*. A control pattern is a set of low-level controllable events that are to be disabled in the low-level controller.

We start with a low-level plant with language L_p, and a low-level controller with language $K_{lo} \subseteq L_p$. We wish to implement further restrictions on the performance of the low-level plant to a language $\widetilde{K}_{lo} \subseteq K_{lo} \subseteq L_p$. This is to be done by translating strings from L_p to a high-level language, $L_{hi} = M(L_p)$, which has an alphabet that contains controllable and uncontrollable symbols, $\Sigma^{hi} = \Sigma_u^{hi} \cup \Sigma_c^{hi}$. In this example, each state in the low-level controller is labeled with either a high-level symbol or τ_0. This implicitly determines *M*. Whenever the low-level controller enters a state marked with a high-level symbol, the symbol is added to the high-level translation of the low-level string.

An alternate implementation assigns an upper level transition (or null) transition to each path through the machine. In this way, an upper level transition can be generated for each lower level, or aggregation of lower level events. ARL has implemented its hierarchy in this manner, however these two approaches have been shown to be equivalent.

For control to be maintained, the high-level controller needs to be able to control the virtual high-level plant via the mechanism shown in the figure. This is called *hierarchical consistency*. It will be true when $M(\widetilde{K}_{lo}) = K_{hi}$, which can be verified for the hierarchies constructed in this experiment.

Note that the example is hierarchically consistent because the only controllable high-level event, *B*, can be disabled with its associated control pattern {d,f}, which blocks the low-level controller from entering the state which transmits *B* to the high-level controller. This is done by disabling all of the low-level events (which are controllable) leading to the low-level state marked *B*.

In the example, the high-level controller adds the following requirement to the behavior of the low-level controller: *an event from the set {d,f} can only occur once in any low-level string*. This translates to the high-level requirement that: *the event* B *can only occur once in any high-level string.*

This technique can be used to allow a high-level controller to control more than one low-level controller as shown in Figure 2.2. The events recognized by the high-level controller arise from the synchronous product of the output symbols produced in each low-level controller. More specifically, a high-level controller implemented in this way can recognize the source of a given event and control each low-level controller synchronously.

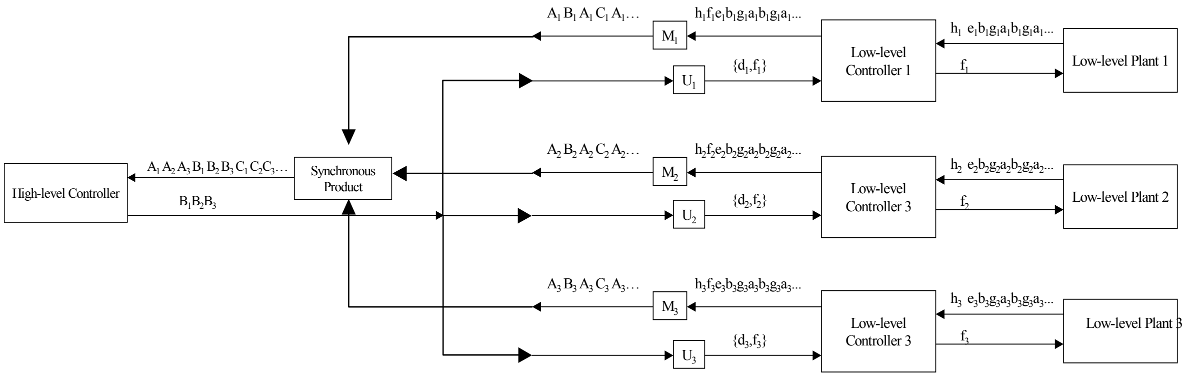

$A_1 B_1 A_1 C_1 A_1\ldots$
$h_1 f_1 e_1 b_1 g_1 a_1 b_1 g_1 a_1\ldots$
M_1
Low-level Controller 1
$h_1\ e_1 b_1 g_1 a_1 b_1 g_1 a_1\ldots$
Low-level Plant 1
$\{d_1,f_1\}$
U_1
f_1
$A_2 B_2 A_2 C_2 A_2\ldots$
$h_2 f_2 e_2 b_2 g_2 a_2 b_2 g_2 a_2\ldots$
High-level Controller
$A_1 A_2 A_3 B_1 B_2 B_3 C_1 C_2 C_3\ldots$
Synchronous Product
$B_1 B_2 B_3$
M_2
Low-level Controller 3
$h_2\ e_2 b_2 g_2 a_2 b_2 g_2 a_2\ldots$
Low-level Plant 2
$\{d_2,f_2\}$
U_2
f_2
$A_3 B_3 A_3 C_3 A_3\ldots$
$h_3 f_3 e_3 b_3 g_3 a_3 b_3 g_3 a_3\ldots$
M_3
Low-level Controller 3
$h_3\ e_3 b_3 g_3 a_3 b_3 g_3 a_3\ldots$
Low-level Plant 3
$\{d_3,f_3\}$
U_3
f_3

Figure 2.2: Hierarchical Control of Multiple Low-level Controllers

The technique can also be extended to form multilevel hierarchies where every non-root node sends higher-level symbols to the level above it, and every non-leaf node sends control patterns to the level below it.

The hierarchical control techniques used in this project will be tested for scalability by proportionately increasing the size of the hierarchy and battle space as shown in figure 2.3, and comparing the results with the objective functions. In 2.3a, a single controller/plane is attacking a single enemy target located in a given area, *A*. In 2.3b, a two-level controller hierarchy with three planes is attacking three enemy targets in an area of *3A*, and in 2.3c, a three-level controller hierarchy with nine planes is attacking nine enemy targets in an area of size *9A*. In addition to the objective functions, the permissiveness of each configuration will be calculated and compared as a function of scale.

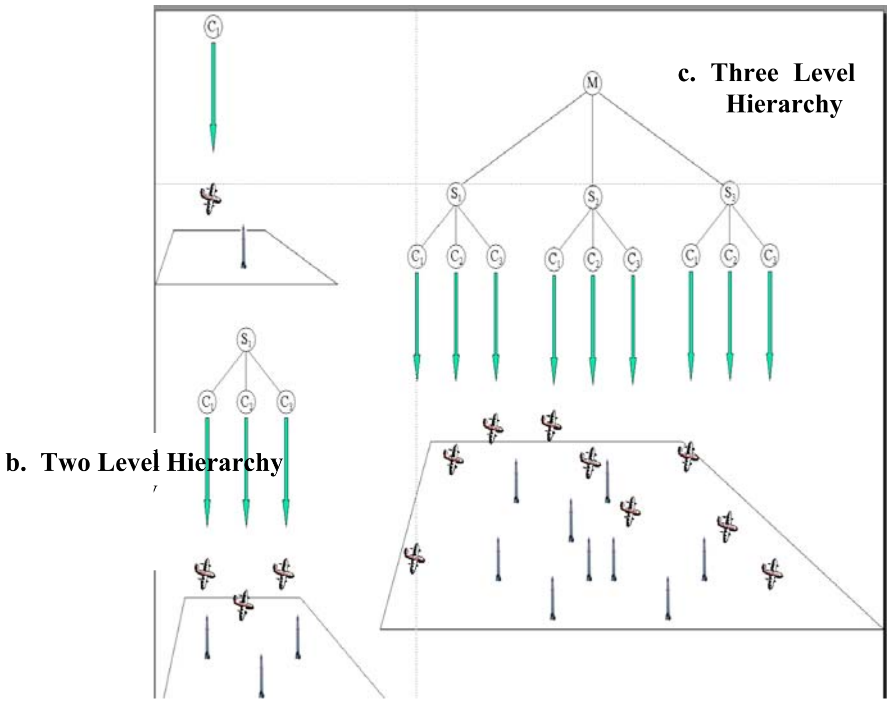


Figure 2.3: Control Hierarchy Scalability Test

Hypothesis C3:
The dynamic hierarchies of interacting controllers developed in this project will improve on the performance of the static hierarchies.

Level of C3:

- Level 1 – Determine the hierarchical consistency and permissiveness for two- and three-level dynamic hierarchies and compare the results with those for the static hierarchy.
- Level 2 – Verify that both hierarchies can control the simulator in a series of runs. Estimate their robustness and permissiveness.
- Level 3 – Run a set of simulations with the controller and the *actual plant*. Verify that the controller operates the actual plant within a set of specifications without significant deadlocks or livelocks. Determine the "information load" at the supervisory nodes in the hierarchy.

Description C3:
We have developed a hierarchical design strategy that allows for dynamically changing the configuration of the hierarchy while still following the mathematical rules for hierarchical DES controllers. The correctness and effectiveness of this strategy will be tested in Experiment C3.

When the controller for an airplane in a fixed hierarchy enters the *destroyed* state, it must remain there because there are no transitions leaving this state. Dynamic hierarchies are implemented by expanding the meaning of the *destroyed* state to *plane is unavailable*, and adding two new events, *plane is replaced*, and *plane is reassigned*. This is illustrated for an abstracted version of the lowest level controller in Figure 2.4.

The supervisory controller is designed so that it does not send the *plane is reassigned*, or *plane is replaced* event until it has authorized one of these two actions. Otherwise, when one of the low-level controllers is in the *unavailable* state, the supervisory controller acts as though it has one less child in its hierarchy. For example, the supervisor node of a two level controller, requests an airplane from the base when one of its low-level controllers is in the *unavailable* state, and, if the request is granted, enables the *plane is replaced* transition. A three-level controller, whose root node will be called the *metasupervisor*, can transfer an airplane from one supervisor to another by instructing the supervisor of the source to enable the *plane is reassigned* event, and instructing the supervisor of the destination to enable the *plane is replaced* event.

We will look at the objective functions for runs with fixed and dynamic hierarchies when no replacements are available from the base, i.e., when each type of hierarchy has access to the same resources. This will test whether the reassignment capability enhances system efficiency.

One obvious advantage of dynamic hierarchies is that they are able to model reserves. We will use the dynamic hierarchy when reserves are available; we expect a significant increase in performance in this case.

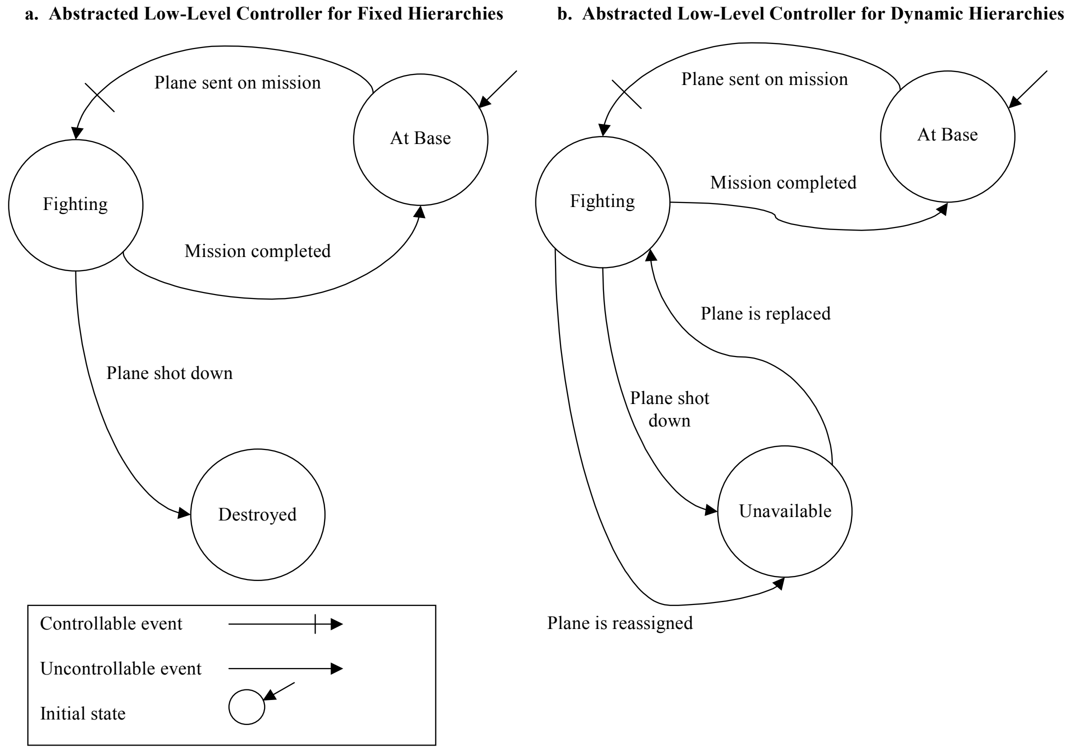


Figure 2.4: Abstracted Low Level Controllers for Static and Dynamic Hierarchies

2.2. Performance Evaluation Experiments

Performance experiments gauge the Discrete Event Dynamic System's (DEDS) ability to perform under various mission parameters; we intend to measure the sensitivity of the controller to various battlefield parameters through Monte Carlo simulations using our air operations simulation. ARL defines the DEDS as the conglomeration of both Finite State Automata control devices and adaptive Tactical Intelligence (TI) algorithms; TI Algorithms augment the DEDS decision process. We hope to show that this amalgam of discrete and continuous control is both suitable and optimal for battlefield conditions.

Hypothesis P1:
The ARL DEDS is capable of sustaining up to 50% information loss without serious performance degradation.

Description P1:
This experiment will measure the effects of sensor degradation on wild weasel performance. Control systems must be able to adapt to imperfect environmental feedback. In this experiment, we obtain a threshold of criticality for information feedback to the DEDS. Sensor quality is varied from within the Air Operations Simulator (AOS) parameters file. Specifically, sensor quality is measured on a scale of 0.0 to 1.0, 1.0 being the highest quality sensor information.

The experiment will be performed by altering the *wild-weasel's* sensor quality parameter in the AOS Startup file. Twenty Monte Carlo runs will be performed with each parameter variation. We estimate these twenty runs will allow error bounds on averages to be kept within 5%. An exact error measure will be computed for each run and used to determine statistical significance of the results. Sensor quality will be varied from 0.0 (worst possible sensors) to 1.0 (perfect sensors). The baseline will use *wild-weasel* aircrafts with perfect sensor information.

The scenario will use four *wild-weasel* aircraft working together to clear two assault corridors in a SEAD scenario as defined by the Cyberland document. SAM sites will be included as targets in the scenario. These sites will be impotent target sites, similar to those used in simple war games. Effects of sensor degradation on plane survivability will be explored in later experiments. A two-level hierarchy of planes and supervisors will be used. The supervisory level will be inactive in the C2 system. That is, they will only act as information relays, not as C2 nodes. Two planes will operate within each corridor. They will operate in two separate regions within the corridor. This will prevent strict cooperation and allow us to ascertain the effects of sensor degradation on the system. Later experiments will allow cooperation between the planes.

Hypothesis P2:
The Two Level ARL DEDS is capable of sustaining up to 50% information loss as defined above without serious performance degradation.

Description P2:
This experiment will measure the effects of sensor degradation at the supervisory level on *Wild Weasel* performance. Control systems must be able to adapt to imperfect environmental feedback. In this experiment, we obtain a threshold of criticality for information feedback to the DEDS in the middle of the hierarchy.

Sensor quality parameters for supervisor planes will be varied in the AOS startup file. The resultant effects on mission performance will be observed. Ten Monte Carlo runs will be performed with each parameter variation. We estimate these twenty runs will allow error bounds on averages to be kept within 10%. An exact error measure will be computed for each run and used to determine statistical significance of the results. Sensor quality will be varied from 0.0 (worst possible sensors) to 1.0 (perfect sensors). The baseline will use supervisor aircraft with perfect sensor information.

The scenario will use two *wild-weasel* aircraft working together to clear one assault corridor in a SEAD scenario as defined by the Cyberland document. SAM sites will be included as targets in the scenario. These sites will be impotent target sites. Affects of sensor degradation on plane survivability will be explored in later experiments. A two-level hierarchy of planes and supervisors will be used. The supervisory level will be active in the C2 system as a sensor relay. The supervisor is equipped with a longer-range radar system. It will relay target-tracking information to the *wild weasels* as they proceed through the corridor.

Hypothesis P3:
The controller will be able to function adequately with up to a 50% loss of target status information[1].

Description P3:
This experiment will measure the effects of status recognition degradation *wild-weasel* performance. Control systems must be able to adapt to imperfect environmental feedback. In this experiment, we obtain a threshold of criticality of target status information for the DEDS. Sensor quality is varied from within the AOS parameters file. Specifically, sensor quality is measured by the probability of ascertaining the status of a target incorrectly. Currently, target status can be undamaged, damaged, or destroyed.

The experiment will be performed by altering the *wild-weasel* planes' state recognition quality parameter in the C4I Startup files used to operate the AOS. Twenty Monte Carlo runs will be performed with each AOS file. We estimate these twenty runs will allow error bounds on averages to be kept within 5%. An exact error measure will be computed for each run and used to determine statistical significance of the results. The probability of an incorrect state estimation will be varied from 0.0 (best recognition) to 1.0 (worst recognition). The baseline will use *wild-weasel* aircrafts with perfect sensor information.

The scenario will use four *wild-weasel* aircraft working together to clear two assault corridors in a SEAD scenario as defined by the Cyberland document. SAM sites will be included as targets in the scenario. These sites will be nilpotent target sites, similar to those used in war games. Affects of sensor degradation on plane survivability will be explored in later experiments. A two-level hierarchy of planes and supervisors will be used. The supervisory level will be active in the C2 system as a sensor relay. The supervisor is equipped with a longer-range radar system. It will relay target-tracking information to the *wild weasels* as they proceed through the corridor.

Hypothesis P4:
Two-level controllers will outperform single level controllers. Dynamic two level controllers will out perform semi-static and static two level controllers.

Description P4:
This experiment will measure the effect SAM "accuracy" has on system performance. Performance will be measured under three DEDS systems: one-level and semi-static two-level and dynamic two-level. Upper-level controllers act as stabilizers for inherently chaotic low-level systems. We will attempt to measure this stabilizing effect. One level control systems have not upper level supervision. Semi-static two level controllers have the ability to dynamically replace planes, but the time required to replace them reduces the effectiveness of the dynamics. Dynamic two-level hierarchies can replace planes quickly.

The experiment will be performed by altering the kill probability for the SAM targets in the AOS. Ten Monte Carlo runs will be performed with each kill probability. We estimate these 10 runs will

[1] Target Status Information: Information pertaining to the "health" status of a target.

allow error bounds on averages to be kept within 15%. An exact error measure will be computed for each run and used to determine statistical significance of the results. The probability of kill per plane per assault will be varied from 0.0 (no threat) to 1.0 (perfect accuracy).

The scenario will use 8 *wild-weasel* aircraft working together to clear two assault corridors in a SEAD scenario as defined by the Cyberland document.

Hypothesis P5:
Three-level, dynamic controllers will perform at least as well as two level dynamic controllers.

Description P5:
This experiment will measure the effect of SAM "accuracy" on system performance. Performance will be measured under three DEDS systems: one-level, two-level and three-level systems. Both multi-level systems will use dynamic reconfiguration to adapt to the battle space. We will attempt to demonstrate the stabilizing effect upper level control has on the lower levels.

The experiment will be performed by altering the kill probability for SAM targets in the AOS. Ten Monte Carlo runs will be performed. We estimate runs these 10 runs will allow error bounds on averages to be kept within 15%. An exact error measurement will be computed for each run and used to determine the statistical significance of the results. The probability of kill per target per plane will be varied from 0.0 (no threat) to 1.0 (perfect accuracy).

2.3. Red Blue Experiments

Hypothesis RB1:
DEDS controller hierarchies are capable of competing successfully against intelligent opponents.

Level of RB1:
- Level 1 – Red SAMs defend the corridor using a strategy based on random movement.
- Level 2 – The Red forces scatter from fixed centers.
- Level 3 – Red forces move in small teams, always maintaining a minimum distance between each other.

Description RB1:
Controllers are synthesized for the mobile SAMs that make up the Red forces. Hierarchies of Red forces are possible. The fact that SAMs are mobile supports their ability to execute intelligent strategies to counter Blue's corridor clearing mission. We have created a hierarchy of three strategies for Red. These strategies embody generic C^2 problems. In strategy 1, Red forces move at random. ISR readings are made periodically, providing Blue with some knowledge as to Red's location. In strategy 2, Red scatter outwards from a source in an attempt to bait blue forces into a false engagement. In strategy 3, Red forces move in small teams in order to increase their lethality.

We have already designed risk-adverse routing strategies as described in section 5.5 of this document. Figures 5.22 and 5.23 illustrate the concepts used in the existing algorithms. We define strategies for blue by updating these algorithms. In each scenario, ISR information is provided giving the position of Red forces at a known point in time. Since Red forces move, Blue can construct probability distribution functions (PDF) that vary over time as to where Red might be. For strategy 1, the PDF will be a normal distribution. Using the time-varying PDF's, risk values can be calculated for regions in the corridor. Blue then tries to construct a route that maximizes the expected value of the number of Red forces destroyed. This expected value contains a term expressing the fact that when Blue is destroyed it can no longer engage Red forces. Each Red strategy defines a new PDF. The PDF determines the risk of taking a trajectory. In this way, Blue strategies are defined by the perceived red strategy.

2.4. Value of Experiments to the DoD

2.4.1. Controller Validation Experiments

Experiment C1:
The first stage in determining the value of applying Discrete Event System (DES) control theory to JFACC command and control is to determine that controllers can be designed with measurable characteristics in terms of the theory itself. Since DES control theory is a relatively new area, this will include original work. Much of this work is motivated by analogy from Continuous Control theory, which has an extensive history in the control of large, complex, dynamic systems. When applied to a single controller, the results will serve as a baseline for the analysis of hierarchical networks of DES controllers.

Experiment C2:
This experiment will determine whether a static network of DES controllers can supplement traditional methods of command and control. The results will determine whether the hierarchy can control and coordinate the lower level entities, which correspond to individual aircraft. The experiment will determine how the hierarchy performs in terms of the control theoretic measurements.

Experiment C3:
This experiment will determine whether a dynamic hierarchy, which allows the hierarchical structure to change in reaction to events on the battlefield, has advantages over a static hierarchy. Quantitative results can be obtained by comparing the control measurements for static and dynamic hierarchies.

2.4.2. Performance Evaluation Experiments
Experiment P1:
Control systems must be able to adapt to imperfect feedback at the lowest level. This experiment validates the notion that discrete event control systems are capable of sustaining moderate levels of incorrect or non-existent feedback.

Experiment P2:
Control systems must be able to adapt to imperfect feedback at the intermediate levels of the hierarchy. This is especially true if information is propagated both up and down the hierarchical system. This experiment validates the notion that discrete event control systems are capable of sustaining moderate levels of incorrect or non-existent feedback.

Experiment P3:
Imperfect information can plague the outcome of C2 operations. Pilots instructed to continue attacking a target until destruction is assured are shown to be inefficient under low information conditions. This suggests important design specifications for the control system and for C2 systems in general. Lower level autonomy could prevent systems from reaching deadlock.

Experiment P4:
Target lethality can alter the outcome of a battle. Here we show that multilevel coordination accompanied with dynamically adaptable hierarchies is better suited to handle threats.

Additionally, we show that over all these systems perform with more stability and are more resistant to deadlock than non-coordinated systems.

Experiment P5:
Target lethality can alter the out come of the battle. Here we show that multi-level, dynamic coordination, accompanied with a dynamic hierarchy is best suited to handle threats. Overall system performance can be improved in this case when resources are low, as supervisors can share planes across portions of an assault corridor.

2.4.3. Red Blue Experiments

Experiment RB1:
The experiment definition provides three generic strategies for Red:

- Random movements—This expresses Blue's uncertainty as to the enemy's movements.
- Scatter away from a point—This expresses enemy movements similar to those experienced in the US's recent attacks on Kosovo.
- Create ambushes by baiting the enemy into a position and attacking *en masse*—Models an intelligent commander in the field.

DEDS controllers need to be able to implement and counteract these basic battle strategies. An open question is whether Blue strategies designed to counteract one strategy will be effective against another strategy. If not, it would be worthwhile to find a method for recognizing the enemy's approach and choosing the correct response.

3. Description of Experimental Setup

Experiments will be run using a war game simulation based on the Cyberland scenario provided by DARPA. A flat terrain is used. Two 4 mile wide corridors are defined from Wewak to Rockatoon city in the Cyberland scenario. 40 F-2E aircraft are allocated to keep the corridors safe; this requires removing all threats within a 44 mile swath (2 mile wide corridor + 20 mile SAM radius = 22 miles for each side). The enemy has 10 fixed SAM sites in the region, 50 mobile SAM sites, 50 AAA batteries and 5 EWCGI radar. Our system controls the activities of the individual F-2E's and coordinates their activities. Figure 3.1 provides a conceptual view of the scenario.
A simple initial approach for preliminary experiments uses a two-level hierarchy. The coordinator assigns regions for aircraft to cover. The individual aircraft choose their own strategies for destroying known targets and patrolling for new threats. As aircraft are destroyed or run out of weapons regions must be reassigned. Similarly, as new threats are discovered regions may be modified. The coordinator is also in charge of coordinating activities during mission initiation and termination.

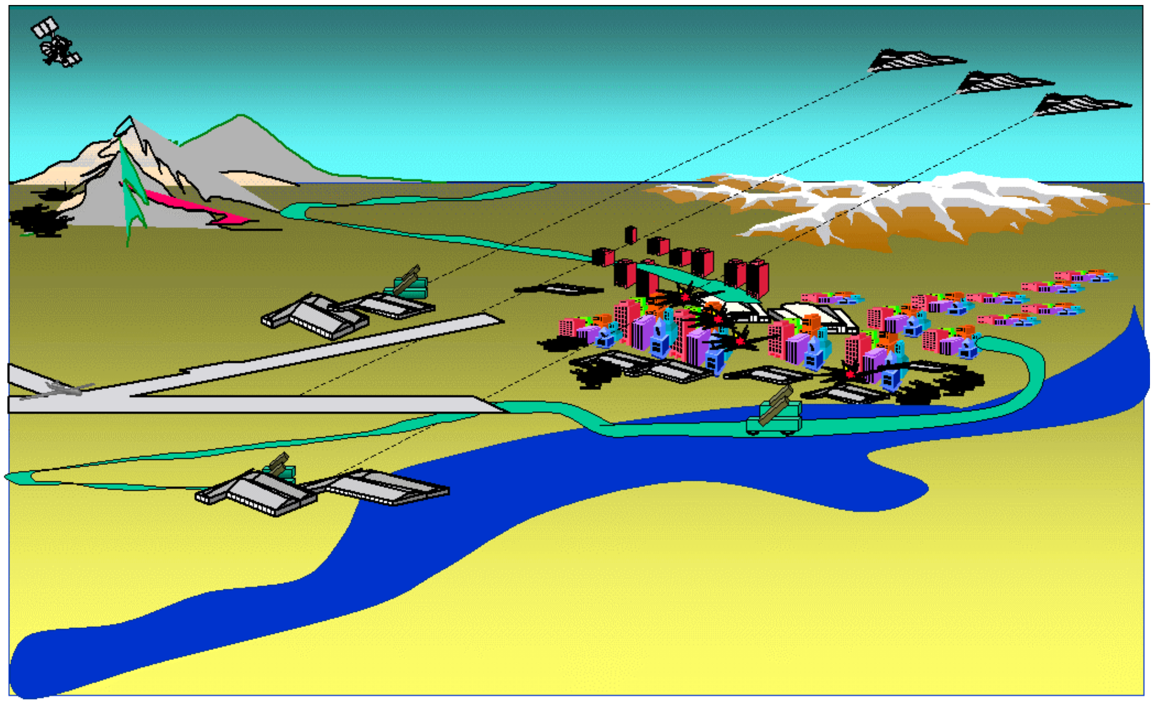


Figure 3.1 Conceptual View of Cyberland Scenario

The setup for planned experiments will facilitate simulation of a limited SEAD scenario. A bombing mission is to be attempted against an enemy airbase. For the bombers to be able to perform the mission, enemy air defenses must be disabled in two corridors leading to the base.

3.1. Simulation Features

For Preliminary Experimentation, we are limiting the number of entities involved. The enemy has three types of entities: (1) fixed SAM sites, (2) mobile SAM launchers, and (3) fixed radar sites. In the initial experiment, mobile SAM launchers are randomly placed. Any target that has been hit is disabled. Friendly forces are limited to fighter aircraft and wild weasel aircraft (that search for and destroy SAMs). Each aircraft has its own on-board discrete event controller. The local controller has access to local information only. The higher-level controller interacts with the system by receiving Intelligence, Surveillance & Reconnaissance (ISR) inputs and sending messages to the aircraft. At first a flat terrain is used and weather effects are ignored in the first iteration.

In the scenario the corridors are given. They are four miles wide at their narrowest, to insure the safety of aircraft flying down the middle of the corridor. Each aircraft starts with an initial mission to be completed. The aircraft's controller determines the decisions that are taken as events occur. Missions will be to patrol parts of the corridor and destroy enemy entities, adjusting the regions covered by each aircraft in response to changing conditions.

We will discuss the features of the simulated experiments; to do so succinctly, we will divide this discussion into four subsections.

3.1.1. Plant

The model of plant dynamics follows a hierarchical structure in which the lowest level (level *n+1*) represents the continuously varying part of the process (e.g. the battle space itself) while the upper levels (i.e. levels 1 to N) represent the discrete-event dynamics resulting from the Command and Control (C2) actions of both friendly and enemy forces. We are developing distributed discrete-event models at levels 1 to N with the assumption that the continuously varying models are implemented by air operations simulators (AOS). Discrete-event plant models follow the standard finite-state automaton (FSA) structure. The rationale for this approach is explained below.

Intelligence of the DEC system increases as each new hierarchical level is added. Upper-level nodes process more abstract information streams, while lower level nodes exchanges more concrete packets of data. As such, lower levels that correspond to decision and control on a relatively fast time scale rely more heavily on local feedback control than upper levels that largely serve as mission planners and top-level decision-makers. Since FSA-based discrete-event control synthesis techniques are fairly well developed and are adequately reported in technical literature, we have adopted the FSA modeling methodology.

The (open loop) FSA model at level *m* in an *n* level control hierachy is an aggregation of all closed loop systems at level *j* for all $j > m$[2].The control law at level *m* is synthesized based on this aggregated model and the corresponding control specifications. The complete decision support system will rely on extensive simulation experiments involving all levels as well as information exchange with the AOS.

3.1.2. Plant or System Identification

Atomic C^2 nodes at the n^{th} hierarchical control level of the Cyberland Scenario, for example SAMs, fighter aircraft, Wild Weasels, etc., are modeled as interacting automata. Figure C.1 shows an FSA model of a fighter aircraft. Controllers on-board these entities accept a time-series of observations from the Plant Model in the formal language of the automata, which is a prefixed closed subset of Σ^* where Σ is the alphabet of atomic events of the agents and Σ^* is set of all finite strings of atomic events including the null event ε. The interface between the atomic agents and the AOS Plant is well defined and limited to the expressions in the formal language that cause state transitions in the automata. The alphabet Σ of atomic events consists of both controllable events Σ_c and uncontrollable events Σ_u as shown in Appendix C depicting the alphabet of the fighter aircraft.

3.1.3. Control Signals

The control signals from the on-board controller at this level are expressed as a function $f : \Sigma_c \to \{0,1\}^{\Sigma_c}$ where $f(\sigma) = 0$ disables the controllable event $\sigma \in \Sigma_c$ and $f(\sigma) = 1$, enables the controllable event $\sigma \in \Sigma_c$. Control efforts are concentrated in the discrete-event part of the process and do not include the continuously varying component at the lowest level. This technology uses multi-level optimization and feedback control. It is supported by extensive simulation. To achieve this, we have adopted a distributed control technology that provides the flexibility to work with multiple, heterogonous plant models and complementary friendly or hostile control systems designed by other research groups. This interface is provided through the definition of an event language. This event language can then be wrapped around the external system allowing for seamless integration with the AC2C test bed. An exhaustive list of the events used in this project is found in Appendix C.

[2] Recall: Lower levels have higher numbers, i.e. in an *n*-level hierarchy—the lowest level is level *n*.

3.1.4.State Observation Signals

The state observation signals are concentrated in the discrete-event part of the process and do not include the continuously varying part of the process at the lowest level. Under ideal conditions of perfect communications, the state observation signals consist of the states q_j of the FSA and all atomic events belonging to the event set Σ as shown in Table C-1 in Appendix C. However, under inadequate communication services (possibly due to weather conditions or equipment failures), some of the events may not be observable. In that case, the assumption of complete observation shall not hold and the effects of some of these unobservable events may be masked to the controller.

3.2. Variables or Correlated Parameters

Independent variables that will be varied are: quality of ISR information, and frequency of ISR updates. Enemy Order of Battle (OOB), enemy strategy, friendly OOB, friendly strategy, number of sorties, and number of days maintained are independent variables that will be kept fixed. Other independent variables to vary in future tests include those used in defining strategies: ISR update frequency, thresholds for changing strategies, parameters of the objective function, etc.

The variables in the objective function include:

- Trajectories of the friendly forces
- The area and duration of portions of corridors kept free from enemy threat
- Time to establish the corridors
- Set of enemy targets destroyed
- The risk taken by (or damage inflicted on) friendly forces
- The list of targets (fixed and mobile) with dependencies
- The positions of (fixed and mobile) enemy forces

3.3. Specification of Test Runs

Specifications of test runs will be largely determined by simultaneous consideration of the control system specifications, range of pertinent plant parameters, and the ISR system. The test runs will be specified by setting the following parameters in the simulator:

- Location Parameter: The location parameter identified the geographic space within which the entire area of interest is specified along latitudinal and longitudinal coordinates.
- An emitter may be used as targets of passive sensor (e.g. RF sensors), as the emission component of communication devices or as the mission components of active sensors (e.g. radar). There are two variations in the detection of emitters, that correspond to signal characteristics: (i) pulsed, scanning, directional beam (e.g. radar transmission); and (ii) non-pulsed, non-scanning, non-directional (e.g. cellular communications, jammers). An emitter either does or does not communicate. Communicating emitters have a maximum transmission range and are required to pass information among various systems or platforms (e.g. AWACs, and C2 Node).
- Sensor Parameter: The sensor parameter encompasses sensor modes (1-3) sensor view, sensor timing, sensor detection, sensor error and sensor reports.

3.3.1.Baseline

The baseline will be generated from an ensemble of simulation experiments to provide specifications for predicting the outcome of air operations events. This includes quantitative evaluation of performance and robustness of the distributed discrete-event control system. The baseline will also be used for supporting analytical work, including plant modeling and defining control system specifications, for subsequent control systems design.

3.3.2.Monte Carlo

The stochastic nature of the air operations problem (e.g. random arrival of threats, and enemy engagements with uncertain outcomes, unpredictable weather conditions, and noisy communication channels) requires stochastic analysis. An analytical solution of the stochastic problem is mathematically intractable because the system is highly nonlinear and of large dimensions. Therefore, Monte Carlo simulation is well suited for analysis of this problem. Randomization of input variables may be used to demonstrate robustness. Such an ensemble of input scenarios will allow us to obtain an estimated mean that can be used as a performance cost function for optimization studies. Furthermore, randomization of input parameters would also be used for sensitivity analysis and thus demonstrate a robustness measure of the closed loop control system. It is important that each Monte Carlo simulation experiment is reproducible in the sense it represents a unique sample point. The challenge here is to plan the Monte Carlo experiments in terms of the number of simulation runs, number of perturbed input parameters, and spacings of these input parameters. Unless carefully planned, the relevant data could be missed in spite of a large volume of simulation data.

3.3.3.Initialization

For each simulation run, the initial conditions of the control system and all plant parameters will be initialized by reading appropriate data files that contain the object specifications of the simulated scenario, their current states, and the mission specifications. Additional data will be needed for a set of simulation runs pertinent to a given experiment. An example is given below:

- Random number generator seeds for selected input parameters
- Number of Monte Carlo passes per experiment and respective starting and stopping times for each simulation run
- Lists of selected outputs

The following paragraphs describe experiment initialization.

1. Simulation start–time and duration – AC2C experiment will take place over multiple day time periods. The initial start time is arbitrary in terms of absolute time. One of the significant experiment metrics will be how much time will be required to clear / roll-back enemy air defense systems from identified corridors, and maintain this condition for minimum length of time. Simulation duration can essentially be considered as a by-product of this metric. In effect, the simulation will be executed until the termination condition is satisfied.
2. Random Seed - Simulation control will provide a random seed for initialization of random sequences. Experiments will be repeatable.
3. Initialization – Simulation initialization will include defining locations of red and blue force elements and defining the ISR state. The ISR state will provide blue forces with initial locations of red elements with accuracy dependent on the target type – higher accuracy for fixed elements, lower accuracy for mobile elements. ISR information will also include a percentage (by target type) of forces known. Experiment will be conducted with air assets at base in a pre-hostility condition. We will start experiment after initialization transients disappeared. Forces will be initialized based on prepared allocation of resources to air bases. We will assume a "normal" readiness condition for attack assets.
4. Scripts - Red force elements will move according to pre-defined script. Essentially, we will define red-force element movements based on achieving scenario goals. Mobile missiles will

generally move with attached ground forces. However, one of the elements within simulation is to provide the capability to "intervene" and change the scripted motion. Blue force elements will move in accordance with operational commands provided by the C2 element. These commands will adapt in real time as forces report their status, and new ISR and battle damage data is obtained.

5. Experimental Bookkeeping – Simulation will report perceived force status to blue elements on a periodic and on-demand basis. This information will include data on red force location and status, and blue force location and status. Reporting will also include logistics – weapons available on a platform, and at airbase, fuel available, and damage / availability of aircraft. Simulation will keep track of any aircraft "kills", actual target damage, and actual vs. perceived force locations. The simulation will measure status of all red-force elements within "roll-back" corridor to determine whether AC2C goal achieved yet.
6. Experimental averages, error statistics and goodness of fit estimations for parameter dependence.

3.3.4. Execution Flowchart and Procedures

Figure 3.2 provides a view of the components used in the experiment and their interactions. Figure 3.3 provides a more detailed execution flow chart for the experiment. The experiment can be conducted either entirely by the ARL T3 organization, or can be separated into components performed by T3 and components performed by the EM. The boxed area shown includes those activities that occur within the experiment time loop – and therefore need to be synchronized with wall time.

The initialization process includes elements with human-in-the-loop (HITL). This refers to operator interaction in the placement of assets at the start of the experiment. The operator will decide the location of assets. Alternately, red forces can be placed using combination of interactive and automated process. In the latter, red forces are specified to be in a geographic region. The simulation randomizes their location within the region.

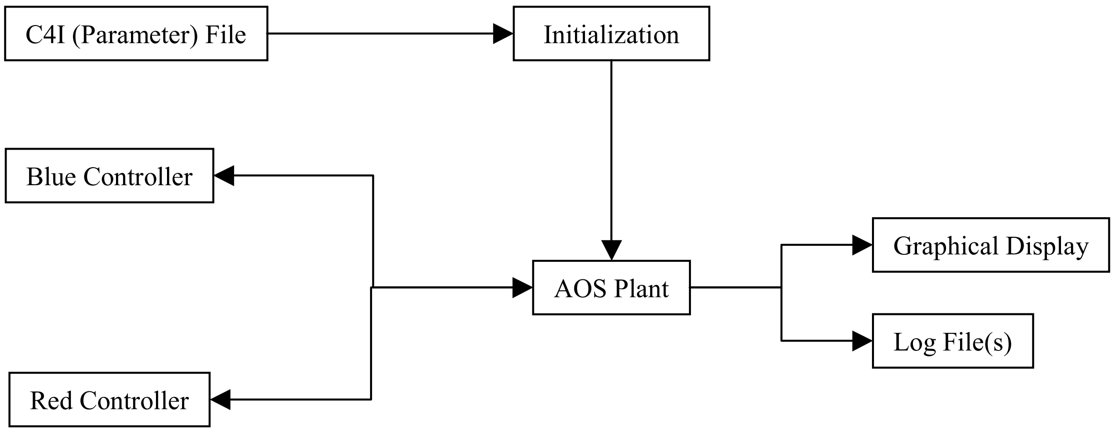


Figure 3.2: High-level Flow of Experiment Execution.

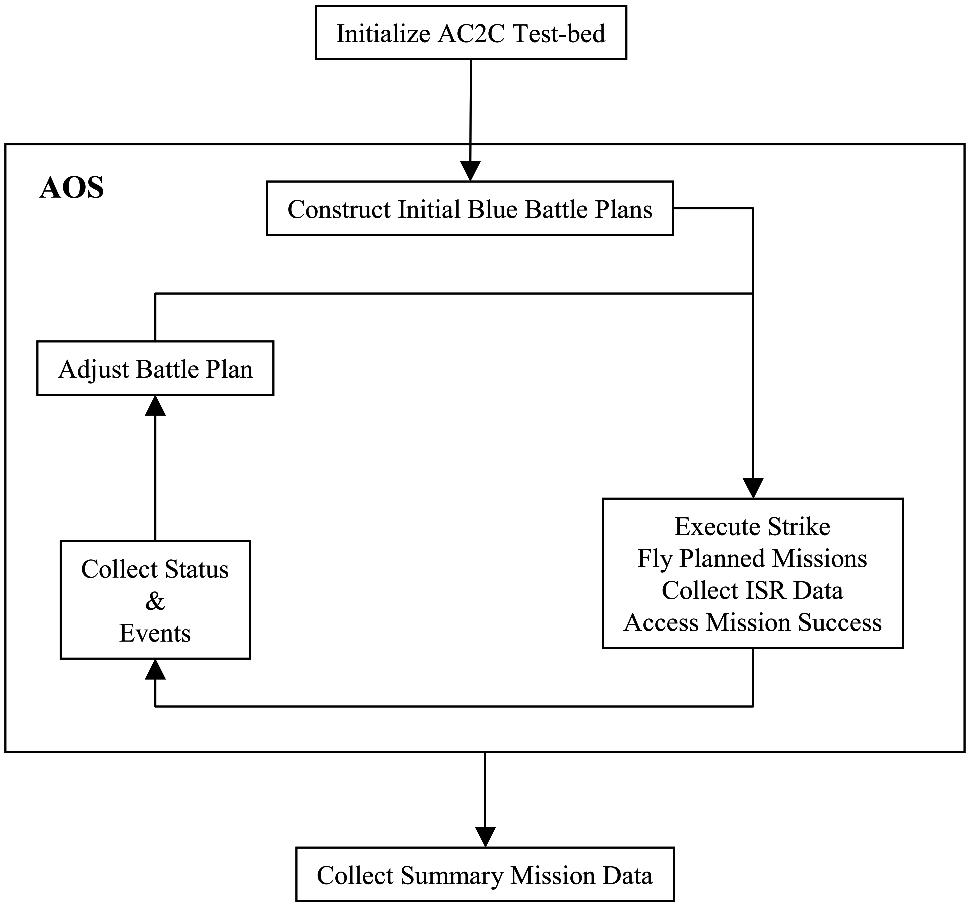


Figure 3.3: Experiment Execution Flow

This is especially appropriate for locations of mobile SAMs and AAA systems – where the important aspect for the experiment is that they are involved and not their precise placement

Blue force route planning can be either computer generated using auto-routing provided by the AOS, or controlled by TI algorithms. The route planner is responsible for the development of the detailed tracks / waypoints for strike elements to attack target defenses. During the pre-attack planning, the generation of the mission plans doesn't need to be conducted inside the time loop since the goal of the AC2C experiments is not to evaluate how-to rapidly mission plan strike routes – but to see how the use of the AC2C controllers contribute to more rapidly achieving the rollback goal.

Inside the time-loop, the plan is executed: 1) aircraft routes are flown; 2) ISR data is collected; 3) target defenses move, detect, and shoot at strike aircraft; 4) Weapons are released by the aircraft; 5) targets are hit and damaged; 6) asset locations and status are reported to the controller; and 7) battle damage information is collected and assessed. Status information is provided to the AC2C controller and a recommended course of action, including re-adjustment of in-flight missions based on updated information, is made. Though not explicitly tested by ARL, future HITL systems are possible in which humans confirm or reject various control recommendations.

3.3.5. Outputs

The output will be a log file with a full list of the events, which occurred during the experiment. We also use a graphic display that allows the results to be visualized.

The experiment will produce a significant amount of "status" data that will be reported to the controller and facilitate adjustment of the battle plan. At periodic instances of time, control elements will "request" status information on: 1) own forces locations; 2) own force health status; 3) own force weapons availability and target assignments; 4) logistics info – weapons and force availability at airbases; 5) ISR estimate of enemy locations; and, 6) Battle damage information – estimate of damage as collected by BDI sensors. The generation of the status information will be in response to both periodic and on-demand CORBA interface calls by the controller.

4. Pre-Lab Analysis

4.1. Predictions

This section summarizes the procedure for analysis and synthesis of discrete-event control systems based on an FSA model of the plant dynamics. Given the plant model and control specifications, the objective is to develop a computer-program that will generate the control algorithm. The major steps for analysis and synthesis of a full observation control law are summarized below:

Steps to Synthesize Supremal Controllable Sublanguage:

1. Formulate $G = \langle X, \Sigma, \delta, x_0, X_m \rangle$ — By hand
2. Write in plain English the specifications (objectives) — By hand
3. Check if G is accessible: $\mathrm{Re}_G(x_0) = X$ — By code
4. Convert the specifications in step 2 into a state machine: $S = \langle Y, \Sigma, \alpha, y_0, Y_m \rangle$ — By hand
5. Check if K=L$_m$(S) is trim: $\mathrm{Re}_S(y_0) = Y$ & $\mathrm{Re}_S(y) \cap Y_m \neq \phi$ — By code
6. Check if K is prefix closed: $\mathrm{Re}_S(y_0) = Y_m$ — By code
7. Check if K is controllable: $pr(K)\Sigma_u \cap L(G) \subseteq pr(K)$: — By code

   Construct $G \| \overline{S} = \langle \overline{Z}, \Sigma, \overline{r}, z_0, Z_m \rangle$ $\overline{S}$ is the completion of S.

   $\forall (x,y) \in Z, \sigma \in \Sigma : \overline{r}((x,y),\sigma)$ defined, $\Rightarrow \overline{r}((x,y),\sigma) \notin X \times \{y_d\}$
8. If K is controllable, Choose S as the supervisory controller.
9. If not, compute the supremal controllable sublanguage: — By code

   $Z_0 := \overline{Z} - Z, k = 0$

   $Z' := Z_k \cup \{z \in \overline{Z} - Z_k \mid \exists u \in \Sigma_u \ s.t.\ \overline{r}(z,u) \in Z_k$

   $Z_{k+1} := Z' \cup \{z \in \overline{Z} - Z' \mid z$ does not belong to trim component of $\overline{Z} - Z'\}$

   Stop when $Z_{k+1} = Z_k$; otherwise $k := k+1$, go to 2

4.2. Validation Criteria

The models are validated based on simulation experiments. Using simulations, appropriate experiments will be designed to validate all hypotheses. The main criterion for validation is the failure to reject a hypothesis under a low confidence level.

The validation criterion for all controllers is that they satisfy the controllability test described above. In hierarchical controllers, each level of the hierarchy must be hierarchically consistent with the level below.

4.3. Evaluation Criteria

The evaluation criteria will be based on metrics such as: the time to clear corridor, percent of time corridors were kept clear, losses incurred, set of enemy targets destroyed, and result of the mission (success, abort, failure).

The evaluation criteria for controllers are their robustness and permissiveness. These quantities are defined in section 5.3 and a method is presented for their calculation. In addition, the utility of hierarchical controllers can be evaluated by estimating the reduction of information flow as a function of control level; it is assumed that good hierarchies will reduce information flow at higher levels.

5. Results & Discussion

This section contains the controller validation and system performance results, a brief summary of how these experiments fit into our overall research program, and how the results will affect (or possibly redirect) future investigations. Algorithmic and theoretical results are also included.

5.1. Controller Validation Results

This section contains the results of experiments to test the validity (in terms of controllability and hierarchical consistency), and evaluate (in terms of robustness and permissiveness) the controllers. It includes experiments for a single controller, a two-level hierarchy, and a three-level hierarchy. There are also experiments to determine how well the plant model (a finite state automaton) represents the actual plant (a full-blown battle-field simulation), and an experiment to determine how well hierarchical control is able to reduce the information flow at the higher levels of the hierarchy.

5.1.1.Experiment C1: Controller for a Single Wild Weasel

Level 1 – Design Controller for a Single Plane:
The plant model for a single wild weasel is shown in Appendix, Figure C.1. The definition of its states and events are shown in Table C-1. The requirements for the single airplane controller are shown in Table C-2. Applying the specifications to the plant model resulted in the controller shown in Figure C.2.

Applying a set of specifications to a plant model can result in a controller with fewer, equal, or a greater number of states. The results shown in the figure are typical in that the specifications resulted in a significant increase in the number of states, from 7 to 29. The original seven states can be seen at the top of the controller graph. The additional states are needed because each physical or logical situation modeled by the plant and controller needs to be represented by a unique state. For example, requirement 5 results in the initial attacking state splitting into 4 states: attacking after 1 alarm has been received, attacking after 2 alarms have been received, …up to 4. The requirements can combine in various ways to result in huge increases in the number of states. This is what motivates us to use hierarchical control.

Level 2 – Verify Controller for a Single Plane:
We then went through the procedure for testing for controllability as shown in Figure C.3 and proved that the controller can control the plant.

As described above, we created a population of plant models near (see section 5.4) the nominal plant model to investigate the controller's robustness with respect to plant uncertainty. Populations were created based on the addition of arbitrary transitions and on the addition of only valid transitions. Examples of valid transitions include:

- Plane is searching for a target; mission is completed or aborted; plane is back at base
- Plane is damaged; plane gets alarm; plane is destroyed
- Plane is damaged; plane gets alarm; plane is damaged.

An example of an invalid transition, which would be excluded from the population in the latter case, is:
•Plane is searching for target; plane gets destroyed; plane is idle in air and safe.

Figure 5.1 shows the fraction of each generation that is controllable by the single plane controller for both the arbitrary and valid transition populations. As shown in the figure, the controllability of each generation drops off quickly for the population based on arbitrary transitions and much more slowly for the valid transition case. The latter case both performs better and is more realistic for practical systems. It was therefore used for all of the robustness experiments in the project.

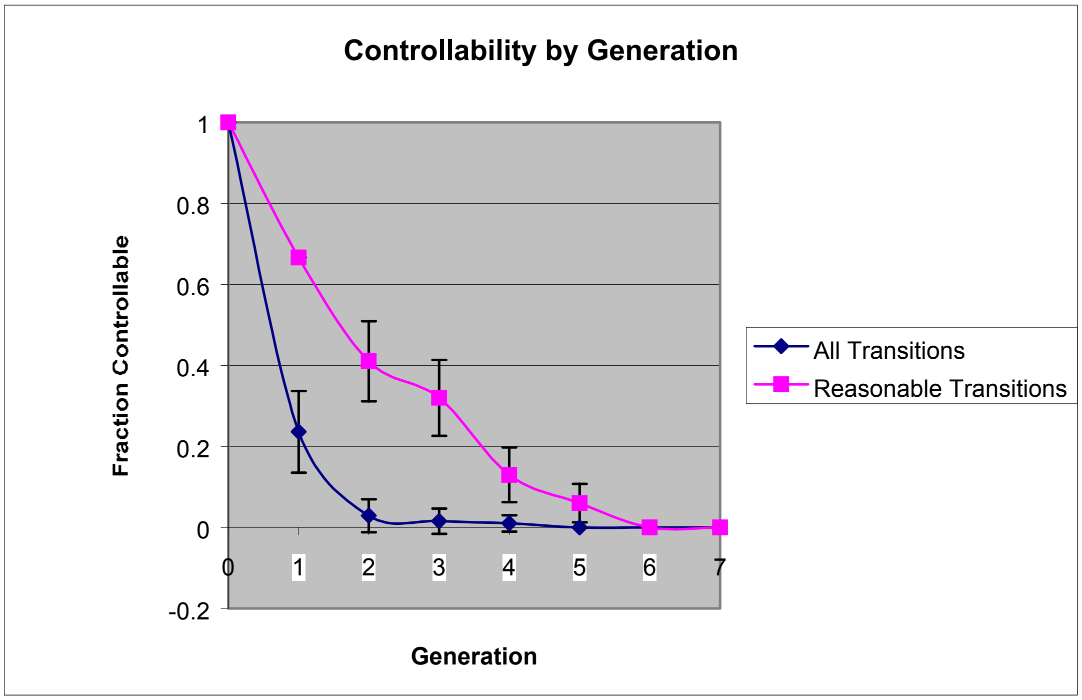


Figure 5.1: Controllability of Plant Model Populations

The robustness for the single level controller is 2.6x10^4, and the permissiveness is 0.86. Figure 5.2 shows the robustness contribution from each generation of the population and the number of automata in each generation.

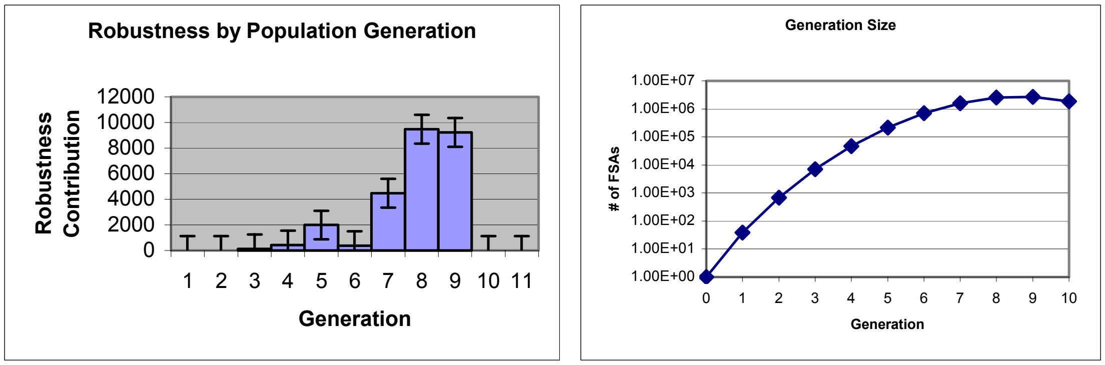


Figure 5.2: Robustness for Single Level Controller

Level 3 – Operate Controller for a Single Plane
Figure 5.3 shows that the plant model, and therefore the controller recognized 99.2% of the events from the actual plant, i.e., the battlefield simulation. 0.8% of the events that are not recognized are caused by search events generated after the lower-level controller has entered an alarm state via an alarm event A. The controller undergoes the following transitions:

$$0 \xrightarrow{s^*} 1 \xrightarrow{A} *$$

Once the controller has reached an alarm state, the search event is no longer an allowable input. However, because of timing issues, an errant search event may be generated after the controller has changed states. The performance of this controller under a number of different scenarios is shown in section 5.2.

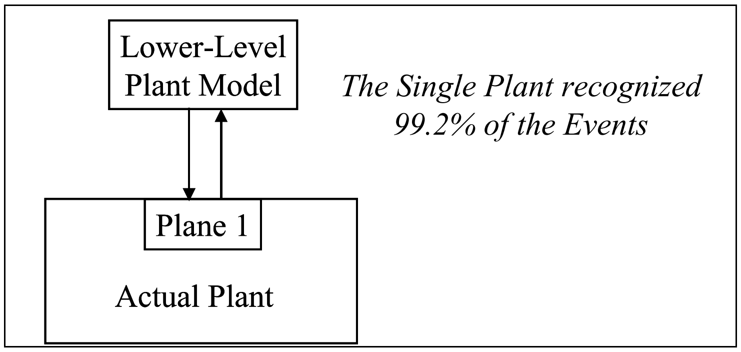


Figure 5.3: Recognition of Actual Plant Language by Plant Model

5.1.2. Experiment C2: Fixed Controller Hierarchy for Wild Weasels

Level 1 – Design a Controller for a Two-Level Fixed Hierarchy
The hierarchy consists of two single-plane controllers at the lower level and a single supervisory controller at the higher level. The same plant models and resulting controllers were also used as the first two levels in the three-level dynamic hierarchy, described in the next section. The only difference is that certain events and their resulting transitions are never enabled in the two-level hierarchy (and therefore cannot occur). They will be ignored for the purposes of this section.

The low level plant is shown in Figure C.4. The only difference from the single-plane controller is that an additional controllable event, S, defined as search for friend, has been added to the plant. Since the new event is controllable, it does not change the controllability of the low-level plant. The idea is that if a plane gets damaged, another, undamaged plane will escort it back to the base. Since the planes do not communicate directly, this is done through a supervisory controller at the next level of the hierarchy. The plant model for the higher-level plant is shown in Figure C.5. The event and state definitions are shown in the left two columns of Table C-3. On the right, is the mapping between events in the lower and higher level controllers.

In hierarchical discrete-event control, the combined nominal plant and lower level controller form the closed-loop plant, which acts as the plant for the higher-level controller.

In order for a high level controller to control multiple low level plants, it must be based on the synchronous composition of each high-level plant model, i.e., each closed- loop plant at the lower level. If there are p plants, each of which has n state, the composition contains n^p states. The controller for two low-level plants, which was used in these experiments, is shown in Figure C.6.

Level 2 – Verify Controller for a Fixed Hierarchy
As described above, we have established that the lower-level controller satisfies the controllability criteria described in Table C.4. The upper level in a hierarchical controller works by restricting the controllable events of the controllers below it. The plant for the upper level is effectively the closed-loop system formed by the lower-level controller(s) and the plant. In order for the upper level to fulfill its specifications, which further restrict the language of the closed-loop plant, the upper level must be able to control the plant indirectly, through the lower-level controller. If this is the case, the hierarchy is said to be hierarchically consistent. Formal criteria for hierarchical consistency are given in the next section. We have used these criteria to verify that the fixed hierarchy is hierarchically consistent.

The bottom two graphs in Figure 5.4 show the controllability of plant populations for both levels of the hierarchy, as a function of generation. The figure shows controllability falls off more slowly, as a function of generation, for the upper level controller. The robustness for the lower level controller is 6.7×10^4, and the permissiveness is 0.87. Figure 5.5 shows the robustness contribution from each generation of the population and the number of automata in each generation of the lower level controller.

The middle graph in figure 5.4 does not go to zero by the tenth generation, as was the case for the lower graph. This is also reflected in Figure 5.6, which shows the robustness contribution from each generation of the population and the number of automata in each generation of the second level controller. Since the robustness contribution increases up the tenth generation, it is too large for us to calculate using our present methods and computer system (a 500 MHz Pentium). We can only say that the robustness is greater than 6.7×10^6. The permissiveness is 0.76.

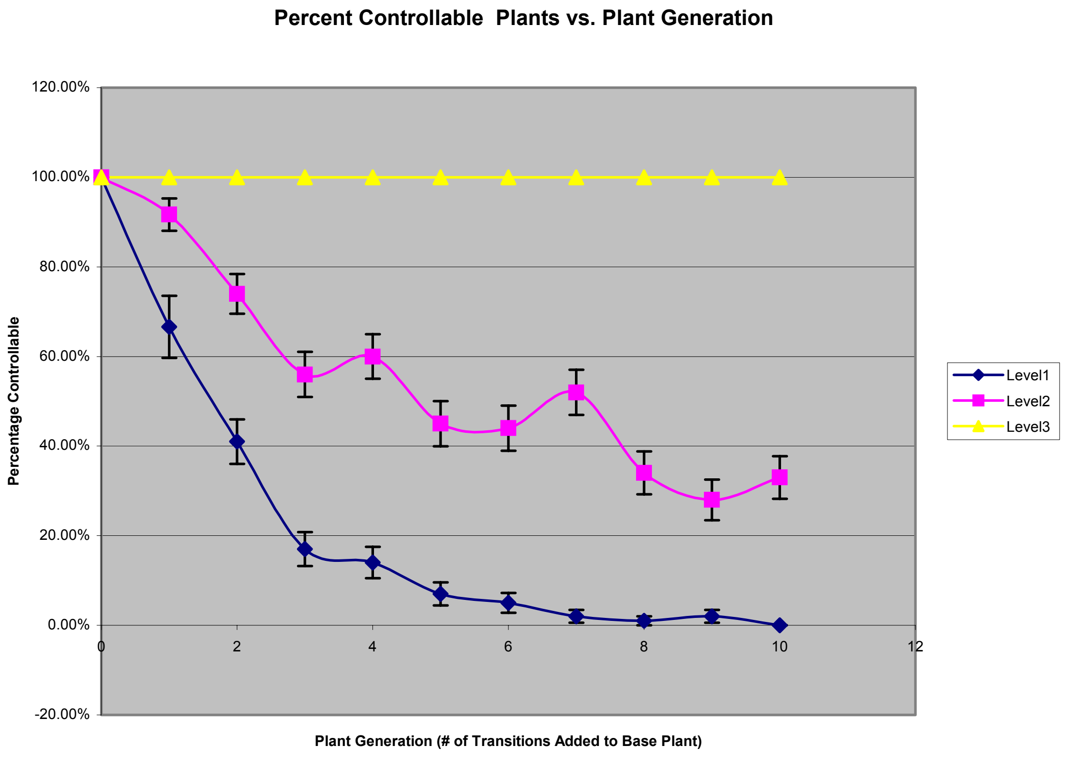


Figure 5.4: Controllability of Each Hierarchical Level Over Plant Population by Generation

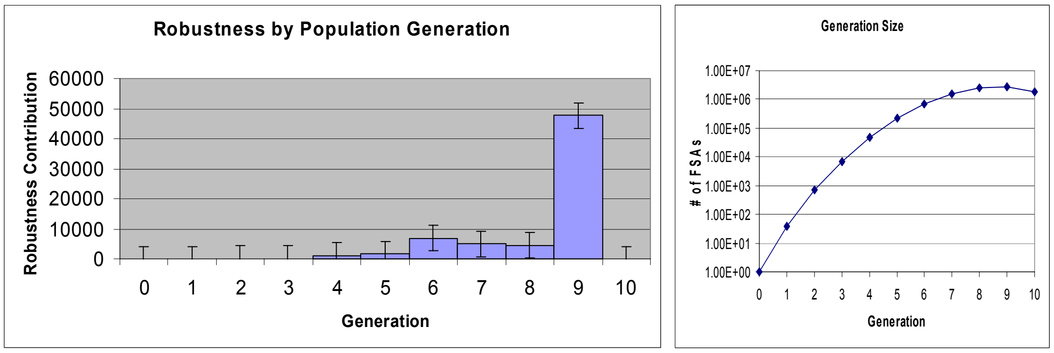
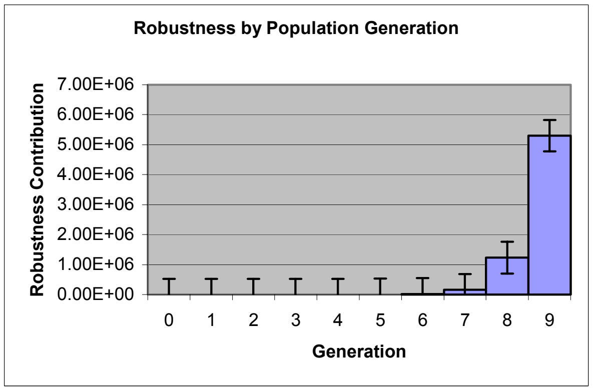


Figure 5.5: Robustness for Lower Level Controller



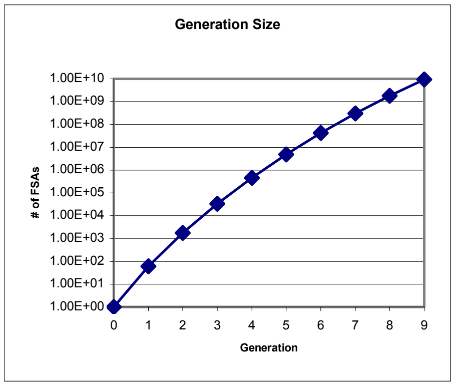


Figure 5.6: Robustness for Second Level Controller

Level 3 – Operate Controller for a Fixed Hierarchy
Our calculations show that the robustness of the second layer is at least two orders of magnitude greater than that of the first. In addition, figure 5.7 shows that the addition of a second layer did not result in any increase in the percentage of unrecognized events during the operation of the controller. These results are evidence that the hierarchical control concept is scalable.

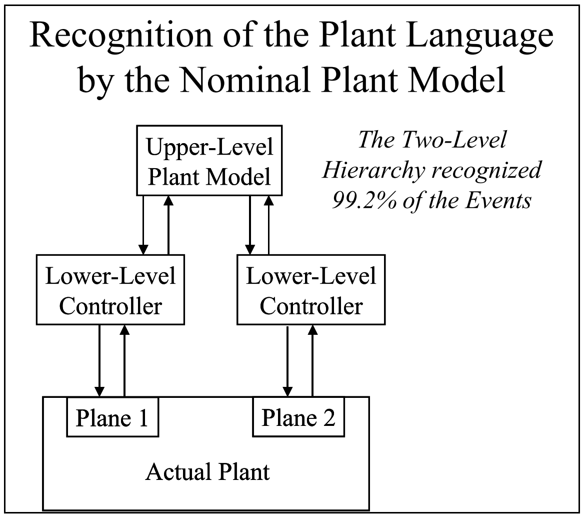

Recognition of the Plant Language by the Nominal Plant Model
Upper-Level Plant Model
The Two-Level Hierarchy recognized 99.2% of the Events
Lower-Level Controller
Lower-Level Controller
Plane 1
Plane 2
Actual Plant

Figure 5.7: Recognition of Actual Plant Language by Plant Model for a Fixed Hierarchy

5.1.3. Experiment C3: Dynamic Controller Hierarchies for Wild Weasels

Level 1 – Design Controller for a Dynamic Hierarchy
Figure C.4 shows the lower-level plant model for both the two-level and three-level dynamic hierarchies. The state and event definitions are shown in Table C.3. The most significant change in the plant is that there are transitions out of state 5, which is now labeled destroyed/unavailable. That is, the transition out of state 5 is interpreted as the replacement of a plane that has been destroyed with another plane. This allows us to implement dynamic hierarchies without violating the assumptions that allow us to establish hierarchical consistency. Planes are replaced by one of two new events, *r* replaces the plane (from another group), and *R* reassigns a new plane from the base. Another new event *x* results in transferring a plane out of the group. Events *x* and *r* cannot occur for the two-level hierarchy because it consists of a single group. It has no way to address other groups to supply or receive planes. The dynamics of the two-level hierarchy consist of losing planes and/or replacing them from the base. The three-level hierarchy; however, can perform transfers from one group to another, which are initiated from the third-level controller. The specifications for the low-level controller are shown in Table C.4.

Figure C.5 shows the second-level controller for both the two- and three-level hierarchies. The states and events are defined in Table C.5. As described in the lower- level controller case, events *X* transfer to another group and *M* replaces the plane; can only occur in the three-level hierarchy. The event *R* reassigns a new plane (from the base) and can occur in both cases. The specifications for the second level of control are shown in Table C.6.

The third-level plant is shown in Figure C.7. Its states and events are defined in Table C.7. The mappings between the second and third levels are shown in Table C.8. The specifications for the third level are shown in Table C.9, and the resulting controller in Figure C.8.

Level 2 – Verify Controller for a Dynamic Hierarchy
The addition of the third level, which allows transfers of planes from one second-level group to another, transforms the fixed, two-level hierarchy into a dynamic three-level control system. We have verified the controllability of the third level and the hierarchical consistency of the controller as described above.

The top graph in figure 5.3 shows the controllability of the plant population for the third level of the hierarchy as a function of generation. It shows that the controllability of all generations is 100%. This is because the only uncontrollable event at the third level is *D*, aircraft destroyed, and there are no valid transitions containing this event that are not already in the plant model. This means the robustness value is extremely large, but finite. All of the valid plants based on the nominal plant model are controllable by the controller. Its robustness is at least $1.9\text{x}10^9$, and its permissiveness is 0.69. Figure 5.8 shows the robustness contribution of the first 10 generations and the number of automata in each generation.

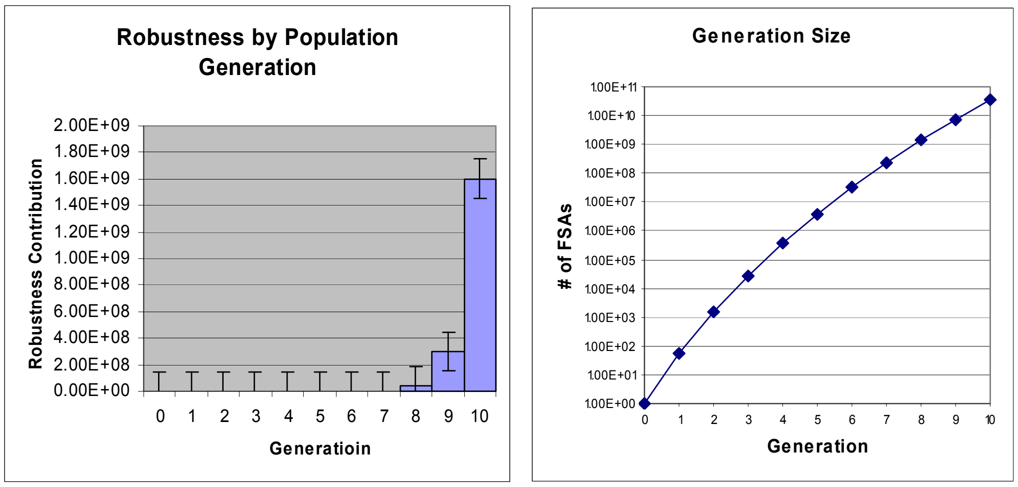


Figure 5.8: Robustness for Third Level Controller

Level 3 – Operate Controller for a Dynamic Hierarchy
Our calculations show that the robustness of the third layer is large compared to the first two. In addition, figure 5.9 shows that the addition of a third layer did not result in any increase in the percentage of unrecognized events during the operation of the controller. These results are additional evidence that the hierarchical control concept is scalable.

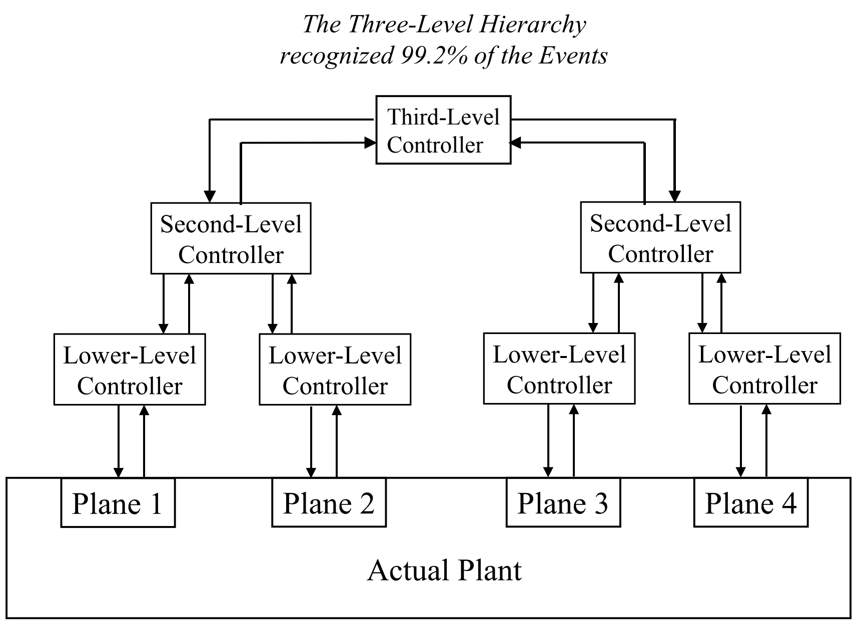


Figure 5.9: Recognition of Actual Plant Language by Plant Model for a Fixed Hierarchy

Figure 5.10 shows the levels of information compression for the three-level hierarchy. The data were taken from a number of runs with a single target. During a run, each aircraft processed 4-7 events at the first level of the hierarchy; each group of two processed 1-3 events at the second level; and the entire group of four processed 1-3 events at the third level. This shows information compression ratios of 5 to 1 between the first and second levels, and 2 to1 between the second and third levels, for a total compression of 10 to 1 for the entire hierarchy. This is evidence that hierarchical control systems are an effective way to hide details from the commander, who may need to control hundreds of planes.

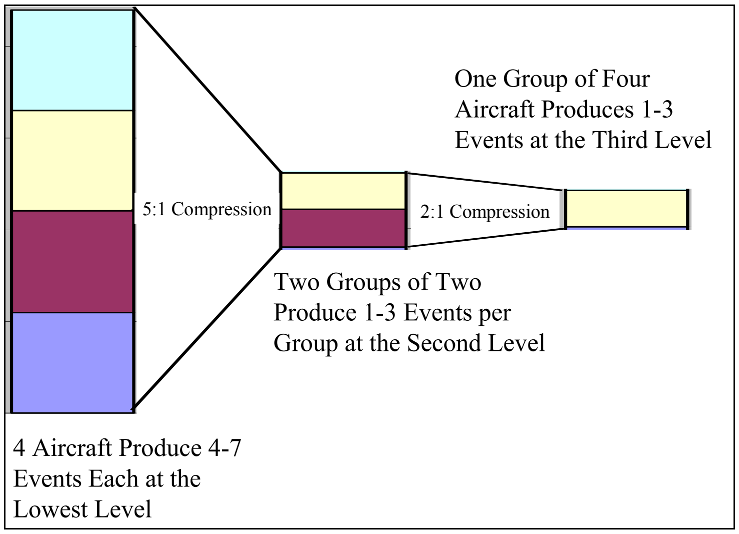

One Group of Four Aircraft Produces 1-3 Events at the Third Level
5:1 Compression
2:1 Compression
Two Groups of Two Produce 1-3 Events per Group at the Second Level
4 Aircraft Produce 4-7 Events Each at the Lowest Level

Figure 5.10: Events per Run for Each Level of the Hierarchy

Figure 5.11 is a plot of robustness vs. permissiveness for the three levels of the hierarchy. It shows the well-known trade-off between permissiveness and robustness, which is evidence of the validity of our analytical methods. It also shows that the higher the level, the more robust and less permissive the controller. The implication is that the controller is more sensitive to the details of the plant model (i.e., less robust), at the more detailed (i.e., lower) levels of modeling.

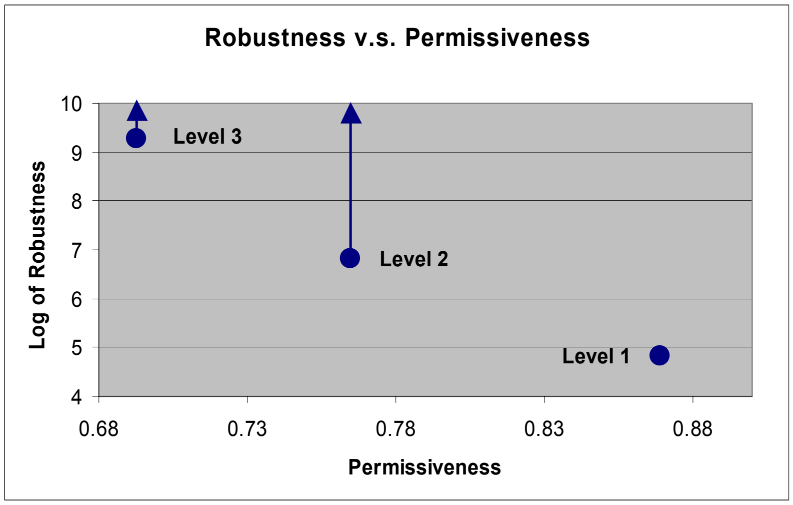


Figure 5.11: Robustness and Permissiveness for Each Level of the Hierarchy

5.2. Performance Results

5.2.1. Experiment P1

Figure 5.12 shows a graphical representation of the results of Experiment P1. As predicted ARL's DEDS performed adequately with up to 50% sensor quality degradation at which point the system became increasingly unstable.

Sensor Quality Degradation (%)	Proportion of Corridors Cleared (10,000 Simulation Sec)	Standard Error
10	0.69375	0.037925
20	0.665625	0.042668
30	0.685938	0.043275
40	0.679688	0.04435
50	0.670313	0.036904
60	0.610938	0.035096
70	0.515625	0.041203
80	0.340625	0.032874
90	0.110938	0.022596

Table 5.1: Experiment P1 Results

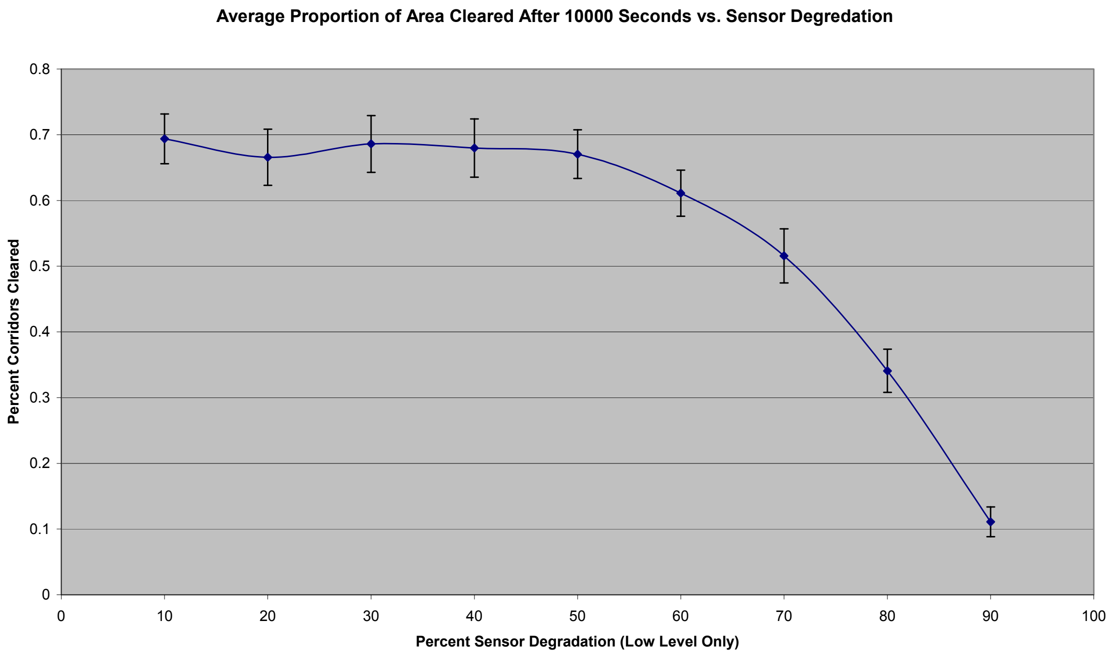


Figure 5.12: Experiment P1 Results

5.2.2. Experiment P2

Figure 5.13 shows a graphical representation of the results of Experiment P2. As predicted ARL's DEDS performed adequately up to 50% sensor quality degradation, moreover the system is less influenced by sensor degradation at the second level. Note performance does not begin to decrease till well above 60% sensor degradation has occurred.

Sensor Quality Degradation (%)	Proportion of Corridor Cleared (10,000 Simulation Sec)	Standard Error
0	0.929167	0.051295
10	0.925	0.052411
20	0.929167	0.052778
30	0.916667	0.052705
50	0.9125	0.053359
60	0.891667	0.055833
70	0.695833	0.108333
80	0.625	0.098601
90	0.4375	0.097222
100	0.4875	0.074587

Table 5.2: Experiment P2 Results

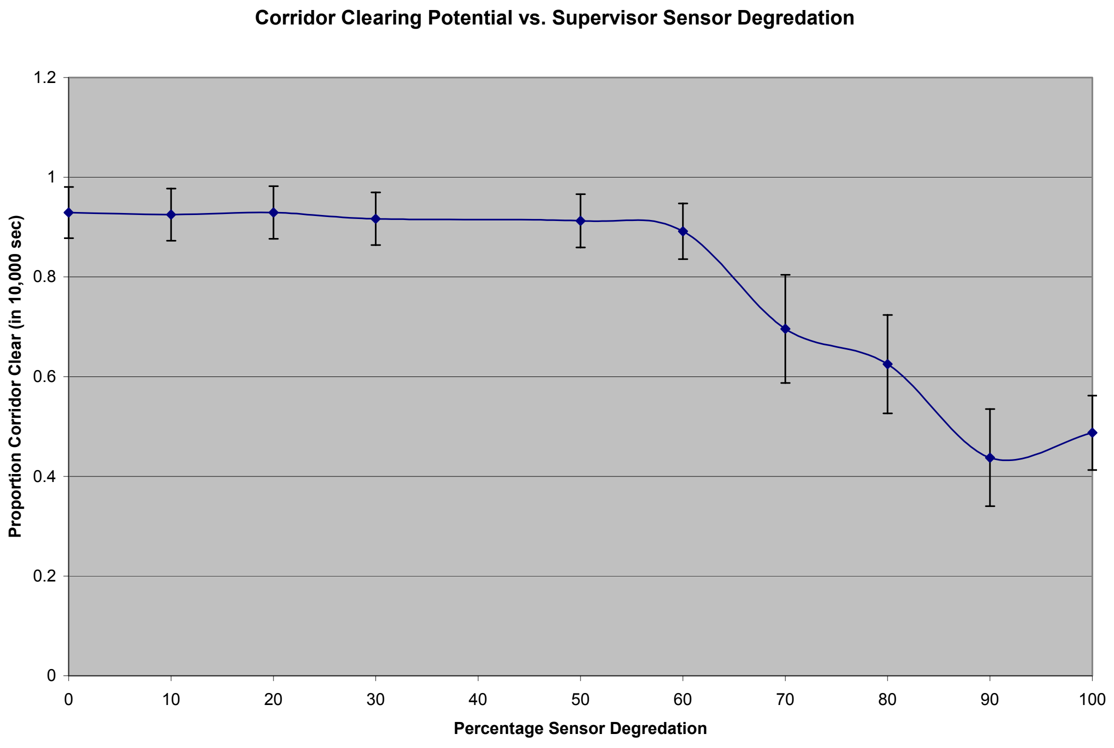


Figure 5.13: Experiment P2 Results

5.2.3. Experiment P3

Figure 5.14 shows a graphical representation of the results of Experiment P3. The figure shows that ARL's DEDS is clearly sensitive to target status recognition. In fact, above a 40% probability of incorrect recognition the controller is not capable of destroying more that 50% of the targets available. This may indicate a problem to be solved in the ARL C2 hierarchy.

State Recognition Degradation (%)	Proportion of Corridor Cleared (10,000 Simulation Sec)	Standard Error
0	0.670313	0.039271
0.2	0.617188	0.030745
0.4	0.489063	0.036877
0.6	0.26875	0.025397
0.8	0.217188	0.01613
1	0.2	0.018014

Table 5.3: Experiment P3 Results

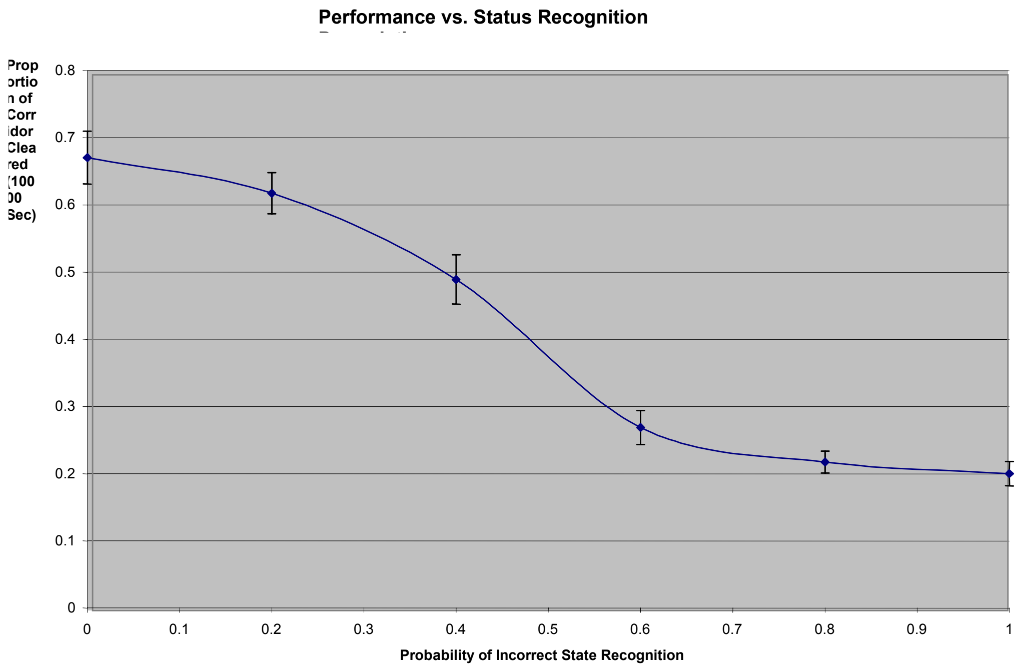


Figure 5.14: Experiment P3 Results

5.2.4. Experiment P4

Figure 5.15 shows a graph of the results presented in Tables 5.4 and 5.5. This data compares non-hierarchical systems to semi-static hierarchical systems. The charts indicate that non-hierarchical systems were able to outperform semi-stable hierarchical systems for weak enemies but faired worse when faced with stronger, more lethal enemies. Figure 5.16 shows a graph of a non-hierarchical system vs. a fully dynamic hierarchical system. Note that the hierarchical system here performs better than the semi-static system from Figure 5.15 and that it is significantly more stable than the non-hierarchical system. This shows the stabilizing force of the second level.

Probability of Kill (Per Assault)	Proportion of Corridor Cleared (10,000 Simulation Sec)	Standard Error
0	0.75	0.055902
0.1	0.65	0.122474
0.2	0.4375	0.16457
0.3	0.33125	0.177805
0.4	0.13125	0.094004
0.5	0.09375	0.075116
0.6	0.05625	0.050861
0.7	0.0125	0.016667
0.8	0	0
0.9	0	0
1	0	0

Table 5.4: Experiment P4 Single Level Controller Results

Probability of Kill (Per Assault)	Proportion of Corridor Cleared (10,000 Simulation Sec)	Standard Error
0	0.26875	0.079167
0.1	0.2625	0.078616
0.2	0.175	0.069222
0.3	0.13125	0.029167
0.4	0.10625	0.0375
0.5	0.1	0.05
0.6	0.05	0.040825
0.7	0.05	0.036324
0.8	0.025	0.027639
0.9	0.025	0.027639
1	0	0

Table 5.5: Experimental P4 Two Level Semi-Static Controller

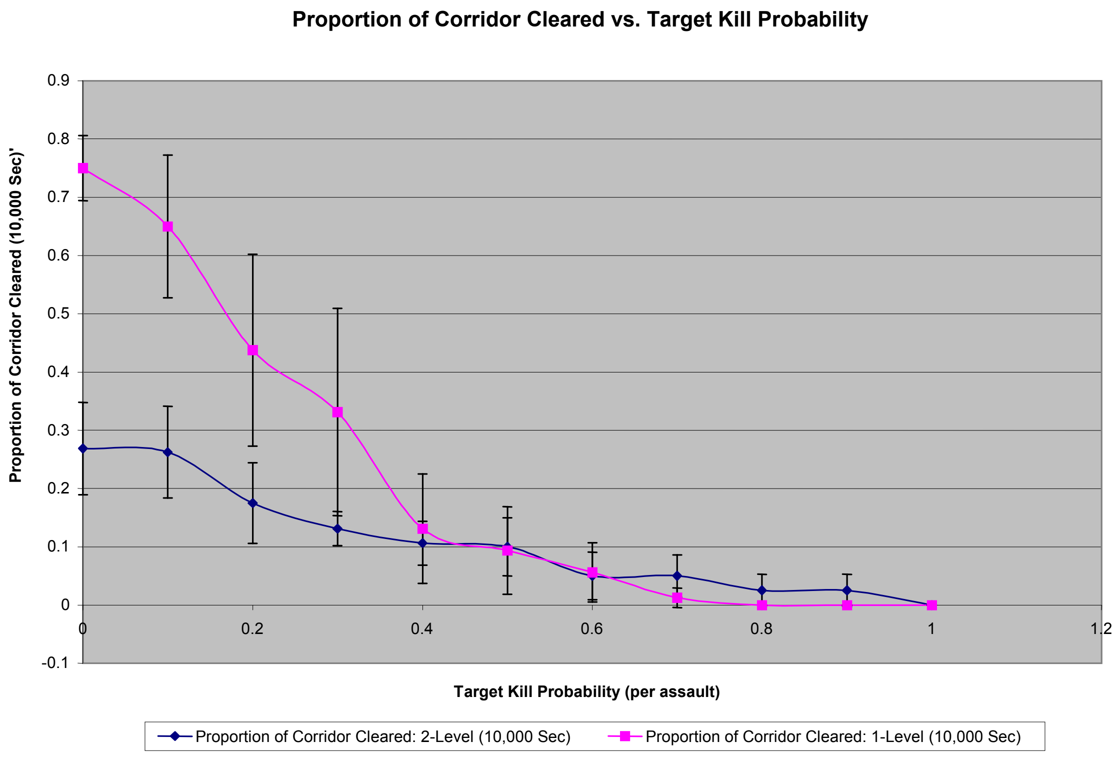


Figure 5.15: Proportion of Corridor Cleared vs. Target Kill Probability

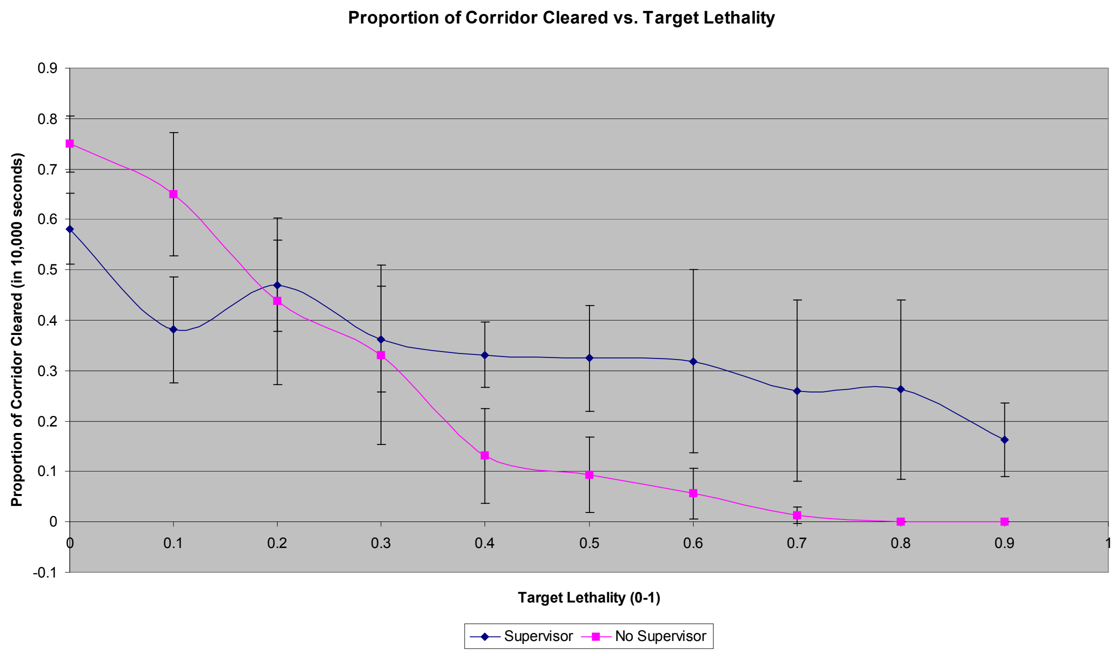


Figure 5.16: Dynamic Hierarchy vs. Single Level Controller

5.2.5. Experiment P5

Figure 5.17 shows the results in Table 5.6. These experiments use a three-level, dynamic hierarchy. We have matched this three-level system against several stationary, hostile targets and varied the lethality of these targets. As the figure demonstrates, there is not a significant difference between the utility of a three-level dynamic controller and a two level dynamic controller. This was expected since the principal use of the third level was dynamic reconfiguration of the second level groups. (i.e. shifting planes between second level supervisors) Even though dead planes could be immediately replaced by planes in the air, at some point a plane would be called from reserves at the base, just as it was in the two-level controller.

Probability of Kill (Per Assault)	Proportion of Corridor Cleared (10,000 Simulation Sec)	Standard Error
0	0.5625	0.07683
0.1	0.5375	0.083749
0.2	0.4	0.089753
0.3	0.4375	0.07683
0.4	0.3625	0.058333
0.5	0.325	0.084984
0.6	0.3375	0.038188
0.7	0.35	0.11667
0.8	0.2	0.076376
0.9	0.1875	0.067185

Table 5.6: Experiment P5 –Three Level Hierarchy

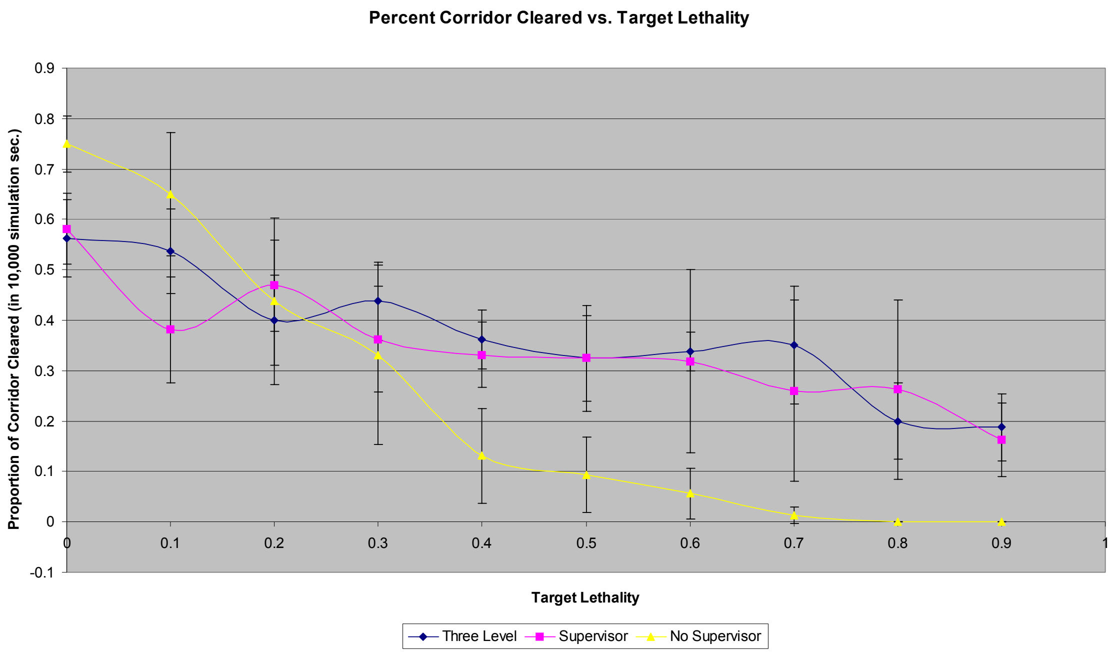


Figure 5.17: Three Level Dynamic Hierarchy

5.3. Red-Blue Experimental Results

5.3.1. Experiment RB1: Level 1

Figure 5.18 shows results from the first Red/Blue experiments. We have matched the dynamic two-level controller from experiment P4 against an adversary capable of making random movements within (and out of) two SEAD corridors. We have varied the lethality of the targets. For comparison, we have provided the results from P4 demonstrating the improved performance the controller has against a moving target. We attributed this performance improvement to the inability of the SAM's to fire while they are moving. Since the SAM's are instructed to move randomly at random times, planes have a non-zero chance of attacking a moving SAM. If this is the case, the SAM's weapons are unable to fire thus improving the likelihood of a successful attack.

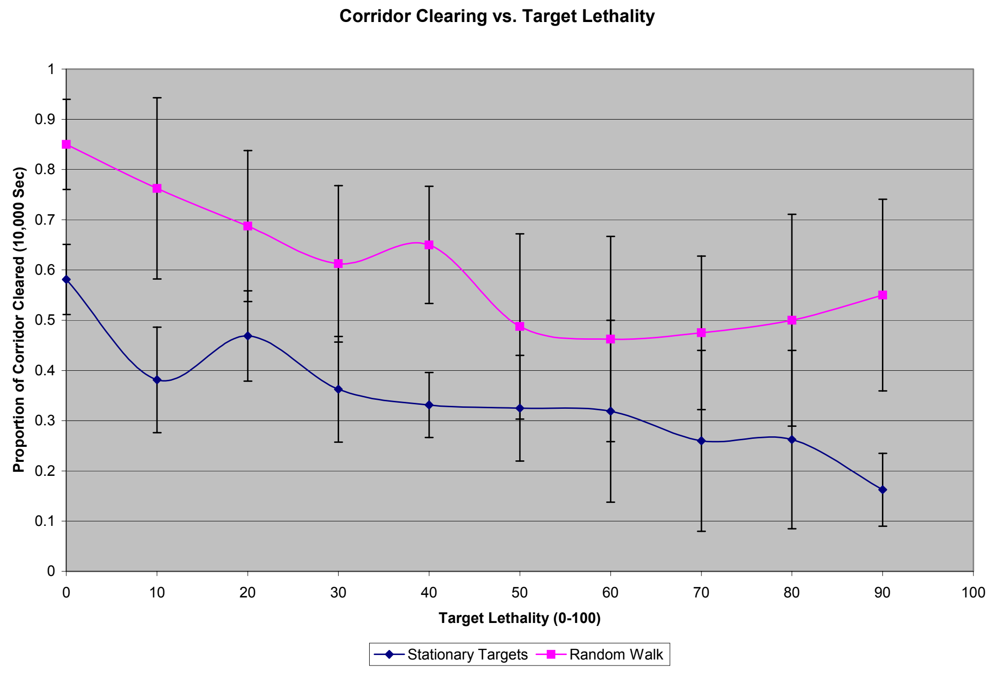


Figure 5.18: Experiment RB1 Results

5.3.2. *Experiment RB1: Level 2*

Figure 5.19 shows the results of RB1-Level 2. This experiment explored the affects of target scatter on battle outcome. Just as in the Level 1 experiment, we see a marked improvement in corridor-clearing performance. This is likely due to limitations in the AOS that make it impossible for SAM's to fire while moving. However, there is a decrease in performance as SAM lethality increases.

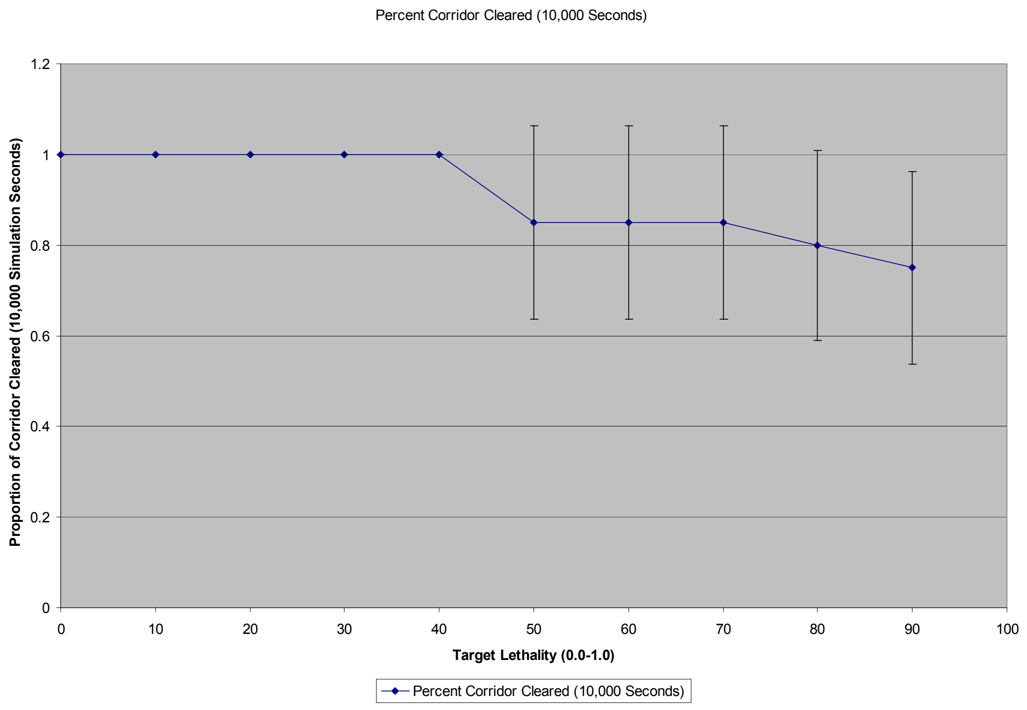


Figure 5.19: Experiment RB1 Results

5.3.3. Experiment RB1: Level 3

Figure 5.20 shows the results from experiment RB1: Level 3; in this experiment, targets where allowed to move in small teams of three. A leader was randomly selected and allowed to move in any direction. The remaining two members of the team would follow the leader maintaining a constant distance. In this way, random target patterns were established. The system performance in this moving SAM experiment was significantly worse than in the pure random walk experiment. We attribute this to an increased probability of encountering a motionless SAM as well as the layout of the battlefield. Targets moving in unison were forced to remain close to each other. This greatly exacerbated the danger to a plane as it attempted to acquire a target. Planes drawn too close to multiple targets can be easily destroyed.

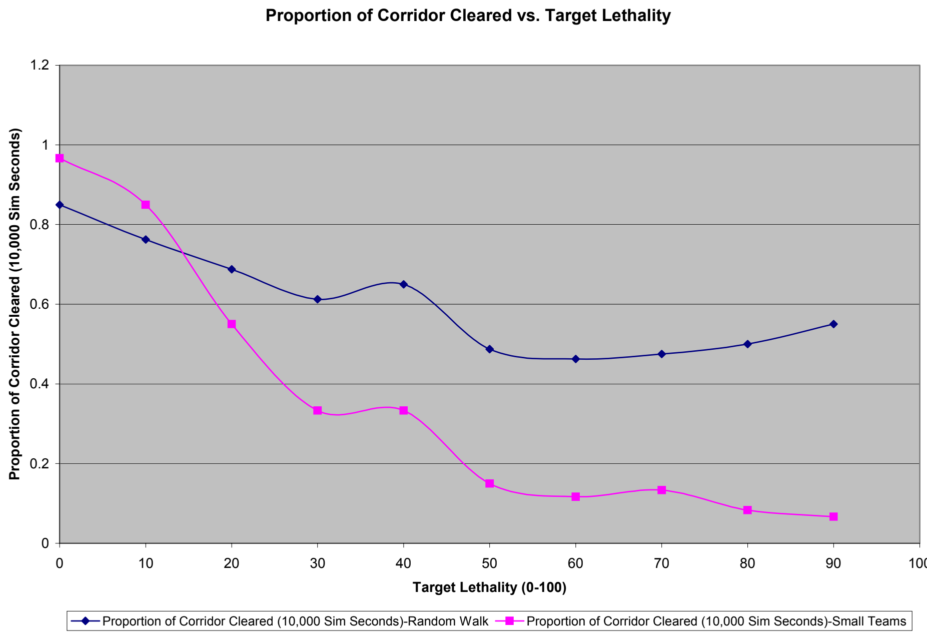


Figure 5.19: Experiment RB4 Results

5.4. Theoretical Results

This section contains control-theoretical results and insights developed during the project. All of the sections represent original work done by ARL except for the section on Hierarchical Consistency that was communicated to us by Dr. W. Murray Wonham. We have derived a measure for formal languages and used it to quantitatively determine robustness and permissiveness for discrete event control systems. This is the first technique for quantitatively evaluating and comparing the quality of these systems. We have also explored an entropy-based (in the information-theoretic sense) technique to suggest effective aggregations for mapping lower to higher-level languages in hierarchical control structures. This is a critical part of designing hierarchical control systems and the approach shows promise.

5.4.1. Language Measure

In order to evaluate the controller, a measure is proposed for formal languages. It will then be used to calculate permissiveness and robustness:

$$\mu(L)=\sum_{i=1}^{\infty} W_i N_i(L)$$

where L is a formal language, $\mu(L)$ is its measure, $N_i(L)$ is the number of strings of length i in language L, and W_i is a weighting factor for strings of length i. For an alphabet containing k symbols, $N_i\left(\Sigma_k^*\right)=k^i$. By convention assume that $\mu\left(\Sigma_k^*\right)=1$, we must find a weighting function such that $\sum_{i=1}^{\infty} W_i k^i = 1$. The weighting function must therefore decrease at least exponentially for large *i*. This defines a family of weighted counting measures that satisfy: $\mu(L_1) \le \mu(L_2)\ \mathit{if}\ L_1 \subseteq L_2$. We define a metric $d(L_1, L_2) = \mu(L_1 \mathrm{Y} L_2 - L_1 \mathrm{I}\ L_2)$ as a measure of the *distance* between two formal languages.

5.4.2. Permissiveness

Assume that a plant *G*, with a language *L*(*G*), and a controller *K* is implemented as an FSA; the language of the controlled plant *L*(*G*|*K*) and assume that: $L(G \mid K) \subset L(G \mid K')$, where, K' is another controller also satisfying the control requirements. It can be said that, for the given application, K' is a better controller than *K* because it allows a richer set of behaviors in the closed-loop system. These additional behaviors could allow the less restricted plant to perform better under some conditions. Given our language measure, we can define a quantitative measure of the permissiveness of a controller *K* on a plant *G* as: $\mathcal{P}_G(K) = \dfrac{\mu(G \mid K)}{\mu(G)}$. The permissiveness will be a real number between 0 and 1.

If the permissiveness is 0, then $L(G \mid K) = \phi$, the controller has shut down the plant and no behaviors are possible. If the permissiveness is 1, then $L(G \mid K) = L(G)$, *i.e.* the behaviors of the closed and open-loop plants are identical and no control is being exerted.

5.4.3. Robustness

Because the true plant can never be precisely known, it is represented by an FSA, called the nominal plant model, which contains all of the available knowledge about the actual plant. The language of the nominal plant model approximates the language of the plant, and the controller is designed on this basis. It is therefore important to determine whether the controller can control languages (i.e., plants) other that the one for which it was designed. This characteristic is known as *robustness* with respect to plant uncertainty. The more robust the controller, the more likely it is to be able to control the actual plant.

To measure robustness, one must look at plants other than the nominal plant model. The measurement is defined in terms of how well the controller can control a population of plants *similar* to the nominal plant model. The method for defining that population is discussed below.

In an applied research problem such as JFACC, insight into developing the plant population can be gained by looking into the physical meaning of the states, events and transitions. The states of the plant automaton correspond to the physical states of the plant and the events correspond to the physical events of the system dynamics. This suggests that states and events should not be varied in determining a realistic plant population. This leaves a finite, but large, population of automata created by varying the transitions. Eliminating transitions that are physically or logically impossible can further reduce the number. For example, a plane cannot attack a target after it has been destroyed. Furthermore, since eliminating transitions merely reduces the plant behavior to be controlled, we have based our population on only the addition of new transitions.

Robustness is defined as $\mathcal{R}_G(K) = \sum_{G' \in S_G} C\left(G', K\right) d\left(L(G), L(G')\right)$, where $\mathcal{R}_G(K)$ is the robustness of controller *K* on plant *G*, S_G is the plant population around plant *G*, and $C\left(G', K\right)$ = 1 if *K* controls G', and 0 otherwise.

In order for this calculation to be practical, the population size must limited without invalidating the results. This is done in two ways. First, the number of transitions that were added to the plant model for a given member can be used to organize the population. We refer to this as the *generation* of the population member. Automata that were created by adding a single transition are referred to as generation 1; automata that were created by adding two transitions are referred to as generation 2, *etc.* As shown in the previous section, there is evidence that the number of controllable automata goes to zero after a small number of generations, and we can cut off the population at that point. Second, we generate a random sample of each generation and use it to estimate the fraction of controllable automata in that generation.

5.4.4. Hierarchical Consistency

Hierarchical consistency can be guaranteed under the following conditions:

- The higher-level controller controls the higher-level plant model.
- There are no silent transitions in the higher-level virtual plant model. This would occur if there were a transition, *w*, in the lower-level machine such that $U: w(s_1,\sigma,s_2) \to x(q_1,\tau_0,q_2)$, and $q_1 \neq q_2$.
- The higher-level virtual plant model is deterministic. This means that there is no pair of transitions, *v*, *w*, in the lower-level machine such that $U: w(s_1,\sigma_1,s_2) \to x(q_1,\tau_a,q_2)$, and $U: v(s_3,\sigma_2,s_4) \to x(q_1,\tau_a,q_4)$ where $\tau_a \neq \tau_0$, and $q_2 \neq q_4$.

5.4.5. Aggregation

In hierarchical control, the higher-level controller operates on an alphabet that is aggregated from the language of the lower-level controller(s). Some symbols in the lower-level language may be ignored, some lower-level symbols may be translated directly into higher-level symbols, and some sequences of lower-level symbols may be translated into a single higher-level symbol. In order for the higher-level controller to effectively control the lower level, the mapping between the lower and higher-level languages must retain important information about the lower level while ignoring the finer details.
Though it is not feasible to automatically aggregate events with no human supervision, certain measurements can be made to reduce the work necessary in constructing an effective event aggregation scheme.

We represent the language of the plant controlled by the lower-level controller as a set of entropy vectors, one for each integer sub-string length. For example, a formal language with alphabet *a,b* would be formally represented as shown in Figure 5.11.

$$V(L) = \begin{pmatrix} e_a \\ e_b \end{pmatrix} \otimes \begin{pmatrix} e_{aa} \\ e_{ab} \\ e_{ba} \\ e_{bb} \end{pmatrix} \otimes \begin{pmatrix} e_{aaa} \\ e_{aab} \\ e_{abb} \\ \vdots \end{pmatrix} \otimes \Lambda$$ [3]

Figure 5.20: Representation of a Formal Language with Alphabet a,b

The value of each component is determined from a finite sample of the language using the formula for modified Shannon Entropy: $V_i \approx \frac{-p_i \log_2 p_i}{N^k_{obs}}$, where p_i is the probability of sub-string *I* occurring in the sample, sub-string *I* is of length *k*, and N^k_{obs} is the number of observed sub-strings of length *k*. Figure 5.12 shows the entropy of the 13 most frequent sub-strings from a simulation run.

[3] Here we intend $\otimes$ to indicate vector concatenation, *not* tensor product.

string	probability p	entropy
a	0.287778	0.0526942
A	0.265556	0.0517624
D	0.197778	0.0471188
DA	0.150575	0.042119
aa	0.143678	0.0411852
Aa	0.14023	0.0407
AD	0.128736	0.0389906
ADA	0.123369	0.0383197
DAD	0.107948	0.0356694
ADAD	0.102941	0.0349094
DADA	0.102941	0.0349094
ADADA	0.0988593	0.0342938
DADAD	0.0963245	0.0337896

Event	Defination
a	fire missle
A	alarm
D	enemy destroyed

Figure 5.21: Entropy of the Most Frequent Substrings in a Simulation

It suggests a mapping from $A^{+}(DA)^{*}D^{+}$ to a symbol in the higher-level language. In the preceding expression, a superscript of + indicates zero or one occurrence and a superscript of * indicates zero or more occurrences.

Using the entropy technique, controller designers can isolate common and rare strings in the plant language and aggregate them to the upper level accordingly. Though it is common for strings with low entropy to be aggregated to the null upper level event, this is not an *a priori* necessity. Often, low entropy strings can signal a rare, and important event in the lower level. It is up to the designer to use the information provided by the entropy calculation.

5.5. Algorithmic Results

5.5.1. Routing

PSU's JFACC team compared the quality and speed of routing solutions generated by different algorithms. The experiments so far have used an minimum distance objective function. A more sophisticated objective function, weighing both risk and time, will be used in future experiments. A description of each algorithm and the results of the comparisons follow.

Greedy Algorithm:
Given a set of enemy targets to destroy and the current location of the wild weasel, a greedy algorithm called the nearest neighborhood search algorithm is developed to obtain a sequence and route to destroy the targets. The greedy algorithm begins with the current location of the wild weasel and destroys the enemy target that can be reached by traversing the least cost path. Then among the undestroyed targets, the algorithm next selects the enemy target that can be reached by traversing the least cost path. This process continues until all known enemy targets are destroyed. Then the wild weasel patrols its assigned region. The main advantage of using the greedy algorithm is to obtain high responsiveness for the tactical intelligence module. Hence the sequence and route to destroy the targets can be obtained extremely fast. However the price to pay for this high-speed response is the quality of the response. In most cases this response from the tactical intelligence is far from optimal, hence the overall objective of minimizing the cost of destroying the targets will not be accomplished.

Resource Bounded Optimization (RBO) Algorithm:
Given a set of enemy targets to destroy and the current location of the wild weasel, an RBO algorithm based on the 2-opt search for the well-known graph theoretic problem, the traveling salesperson problem, is developed. Note that the problem of obtaining a sequence and route to destroy the targets that can be stated as a Hamiltonian path problem as explained in the algorithm that follows. In the RBO algorithm, the solution from the greedy algorithm is taken and improved. During each iteration, a random, pair-wise interchange to the current sequence and route to destroy the targets is performed. After several iterations, the algorithm will

converge to the optimal solution. The number of iterations the tactical intelligence will perform will depend on the time available to respond. The solution quality will improve with the number of iterations. This algorithm can be stopped at any iteration and a solution can be obtained. The numerical examples indicate a vast improvement in the solution quality (as compared to the greedy algorithm and the optimal algorithm) in a few iterations. However, the disadvantage is that it may take a long time to obtain the optimal solution especially in ill-posed problems.

Hamiltonian Path:
The path traversed by the wild weasel from the current location to the last destroyed target is referred to as the Hamiltonian path in graph theory literature. The algorithm uses a network representation such that the known enemy targets are the nodes of the network. There is an arc from every node to every other node denoting the ability to go from any target to any other target. The cost on the arc represents the cost of traversing from one target to another and is computed using the expected travel time and expected risk. The algorithm begins by solving the minimal spanning tree (another well-known graph theoretic problem) of the network. If the spanning tree is not a Hamiltonian path, then it has arcs that violate the Hamiltonian path requirements. The solution of the minimal spanning tree acts as the lower bound. Then at every iteration the lower bound is improved using a branch-and-bound technique where one of the violating arcs of the spanning tree is set to a high cost. The algorithm stops when a Hamiltonian path is obtained, i.e., there are no more branches to consider. This algorithm can guarantee optimal solution after a sufficiently large number of iterations. The major drawback is that if the algorithm is stopped during any iteration, no solution will exist that can be responded by the tactical intelligence.

Optimal Algorithm:
This algorithm uses complete enumeration of the entire solution space to obtain the optimal solution. Therefore every possible sequence and routes to destroy the targets are considered and the best is chosen. This is a very time-consuming algorithm and takes *n!* computations if there are *n* targets to be destroyed. Since this algorithm guarantees an optimal solution, it is used for benchmarking other algorithms.

The algorithms are tested here for different numerical values. In particular, the performance of the new RBO algorithm developed is tested against the other algorithms for the SEAD scenario.

The following table shows the average results of experiments run with thirty sets of enemy targets. In these experiments, there are eight enemy targets and one attacking plane. Resource Bounded Optimization (RBO) is used with vary numbers of iterations, to determine the value of additional iterations compared to their cost in computing time.

Algorithm	Solution Quality (compared to optimal solution)	Number of Floating Point Operations
Greedy (nearest neighbor)	.9570	992
RBO – 5 iterations	.9683	2,589
RBO – 10 iterations	.9727	4,095
RBO – 25 iterations	.9824	8,601
RBO – 50 iterations	.9845	15,898
RBO – 100 iterations	.9868	30,395
Hamiltonian Path (few iterations)	.9907	49,048
Optimal (complete enumeration)	1.0000	351,769

Table 5.6: Average Results of Experiments Run with Thirty Enemy Targets.

One final result noticed by PSU's JFACC team, was the effect of target positioning on a mission's success or failure. We noticed that planes did better with widely spaced targets than with closely spaced targets. Often a plane would attempt to move directly through the field of fire of multiple targets because of its nearest neighbor attack algorithms. We will rectify this by constructing a new assault algorithm and testing it against the old results. Figure 5.12 illustrates this issue.

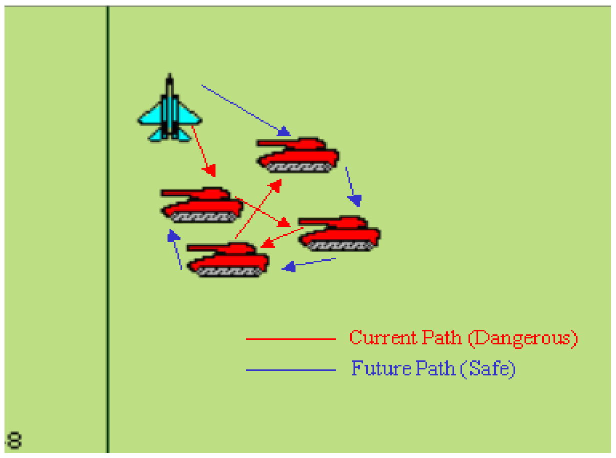


Figure 5.22: High Risk Situation.

The new route optimization problem, minimizing risk *time, is shown below. In the next generation of this model, we are using distance as a proxy for time, and calculating a flight path to follow, instead of a sequence of targets. Figure 5.13 illustrates the new problem definition for attack plane routing.

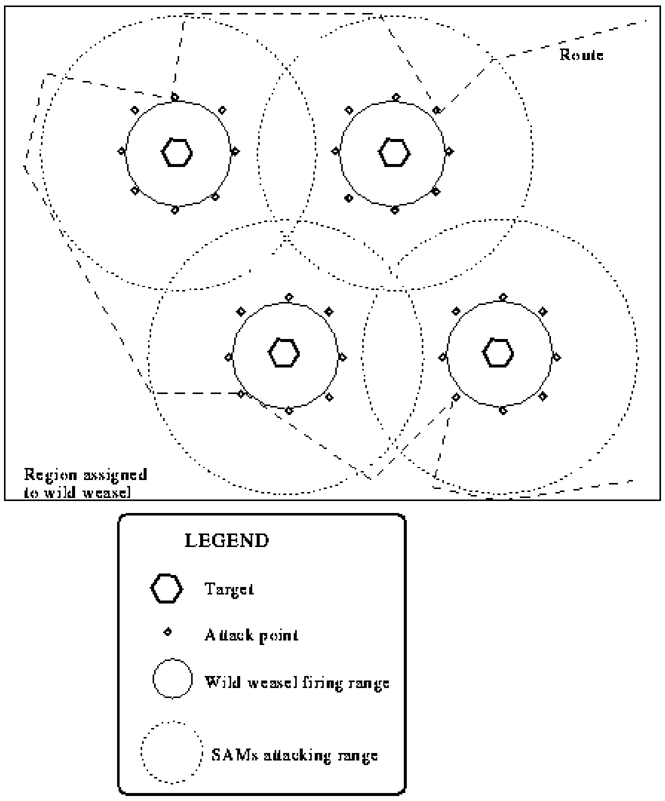


Figure 5.23: Wild weasel platform route chosen to minimize risk.

5.6. Other Experiments

5.6.1. Controller Metric Experiments: Preliminary Results for Plant Uncertainty Tolerance

We have obtained some preliminary experimental results on the sensitivity of our controller with respect to uncertainty in the plant, that we thought may be of interest. We have proposed this as a measure of *permissiveness*. The controller is designed to control a nominal plant model. We have tested it, for controllability of similar models, which are modifications of the nominal plant model. Figure 5.15 illustrates the relationship between the language of the plant model, L_p, the language of a population of similar models, L_{ext}, and the set of arbitrary strings of plant events, Σ_p^*.

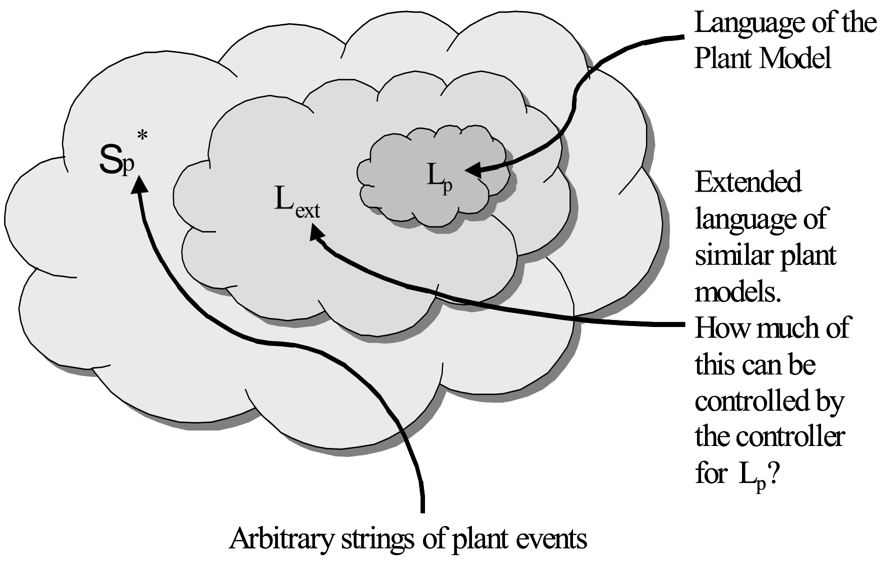

Language of the Plant Model
S_p^*
L_{ext}
L_p
Extended language of similar plant models. How much of this can be controlled by the controller for L_p?
Arbitrary strings of plant events

Figure 5.24: Language Domains for Similar Plant Models

Given $L_p = L(G_p)$, where G_p is the Finite State Automaton (FSA) plant model, and a controller, C, which controls G_p, we can define a set of FSAs, $\mathcal{G}$, where each element of $\mathcal{G}$ is derived from G_p by the addition of a finite number of uncontrollable transitions, and $L_{ext} = \mathrm{L}(\mathcal{G})$.

We estimate C's performance with respect to plant uncertainty as the fraction of the FSAs in $\mathcal{G}$ that are controllable by C. The set $\mathcal{G}$ can be divided into subsets as follows:

$$\mathcal{G} = \bigcup_{l=0}^{l=l_{\max}} \mathcal{H}_l$$

where each subset, $\mathcal{H}_l$, contains the FSAs that can be derived from G_p by adding l uncontrollable transitions, and $l_{max} = n_u n_s - t_{up}$, where n_u is the number of uncontrollable events, n_s is the number of states, and t_{up} is the number of uncontrollable transitions in G_p.

The subset $\mathcal{H}_0$, which contains only G_p, is referred to as *generation 0*; the subset $\mathcal{H}_l$, which contains the plant models derivable from G_p with the addition of a k uncontrollable transitions, is referred to as *generation k for k=1,2.....* Figure D.2 shows estimates of the number of FSA plants per generation, which grow exponentially.

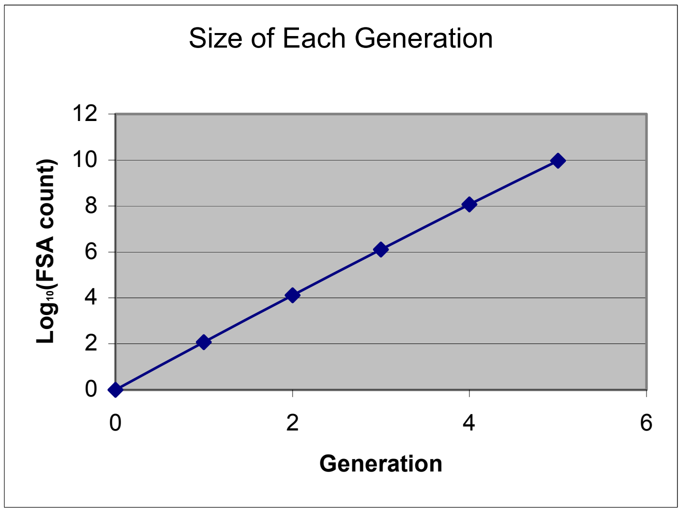


Figure 5.25: Number of Similar Plants Per Generation.

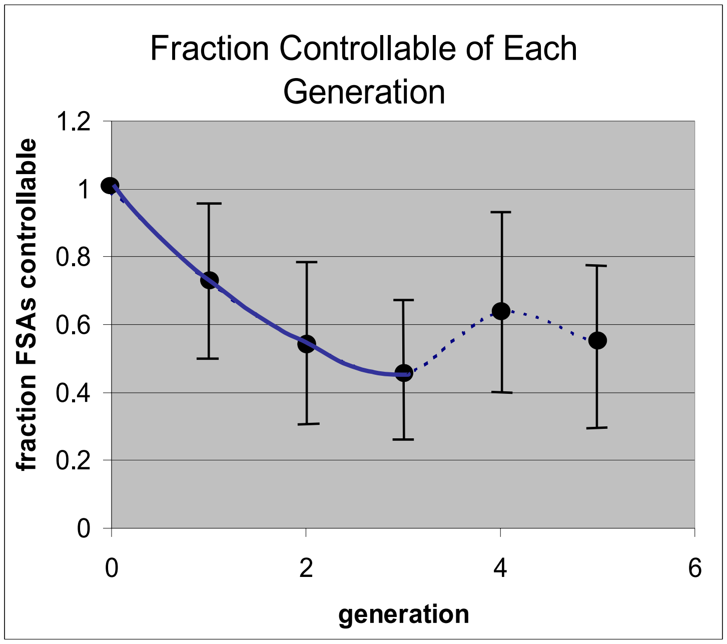


Figure 5.26: The Controllability of Each Generation in $\mathcal{G}$

Figure 5.17 shows estimates, with error bars, of the fraction of each generation that is controllable by C. It controls all of generation 0, the original plant model, and about 75% of generation 1. The fraction decreases until generation 4, where it appears to increase. We have indicated the last part of the graph with a dotted line because the features in this region are smaller than the error bars and may be statistical fluctuations. If the permissiveness is defined by the fraction of $\mathcal{G}$ that is controllable by C, its value will be dominated by the controllability of the last few generations due to the exponential increase in $|\mathcal{H}_l|$ with respect to l. The graph indicates it will be between 40% and 60%.

We will continue the analysis of these results to verify our conclusions and increase the sample size to improve accuracy. We will also be looking at specific plants in the population to gain insight as to why some are controllable and others are not.

5.6.2. Early Performance Experiments:

Our prototype test bed incorporates a simple environment that tracks the status of multiple planes and targets in featureless terrains of varying size. It includes hierarchical information passing and tactical intelligence. We use probabilities of kill from the cyberland scenario and restrict the altitude of the plane to 10,000 feet. We ran each experiment type on varying game field sizes. As the game board gets larger, the probability of successfully destroying the targets decreases. Additionally, the probability of successfully destroying all targets decreases as the number of targets increases.

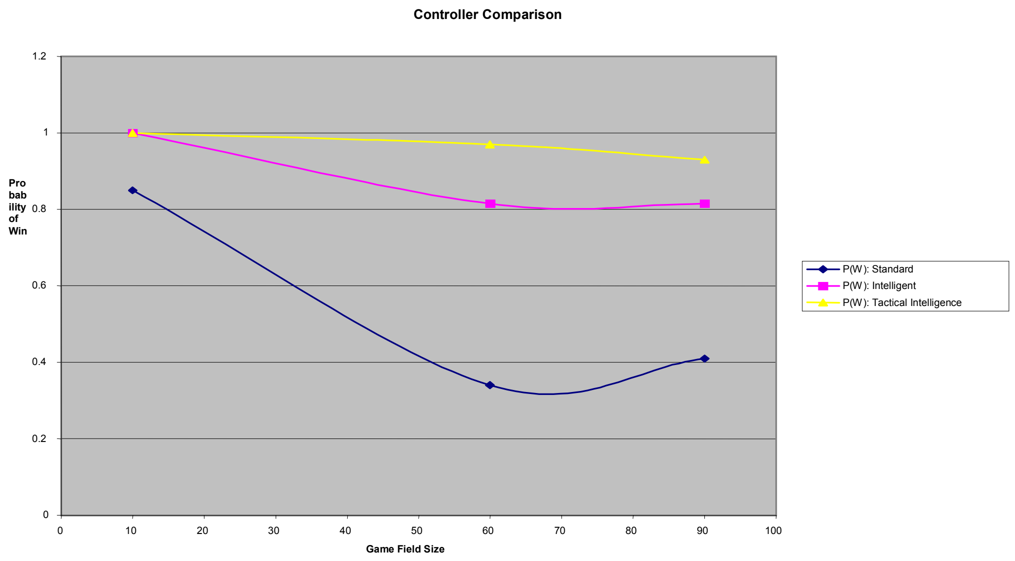


Figure 5.27: Results From Two Aircraft One-Target Scenarios.

Figure 5.18 presents results from a tactical intelligence controller, where two planes initiate a coordinated attack against a single target. The blue line is a graph of the proportion of wins by a set of planes using a normal controller with coordinated attack but not sharing information. The pink line shows the results of incorporating coordination into the plane controller. One plane informs the other when the target is detected. The yellow line allows the planes to make coordinated maneuvers in attacking the target. A higher level of intelligence increases the chances of success. These tests were performed partially to debug our testbed, but also to illustrate the value of coordination in C^2.

SEAD Scenario

ARL's JFACC team has run a variety of experiments with our simulator and current controller. Figure 5.19 contains results from our initial experiments. The scenario has been initialized with one or three planes as labeled. During the three-plane experiments, upper level tactical intelligence allows planes to assist each other during combat. Specifically, if one plane is destroyed the others assume its target list and continue with the mission. During the single plane experiments, we have varied the ability of the controller to observe events as they are generated by randomly masking event generation. As you can see this masking has a negative effect on the ability of the controller to perform; however, statistics show that even if over 70% of the events were to be unobserved there would be a finite probability that the controller could still successfully destroy one target.

Another experiment tests the ability of the controller in unforeseen conditions. We test the controller's reaction to variations in the range at which it is allowed to fire. As is expected the closer a plane needs to be to shoot the better the chance it has of dieing. However once the controller is allowed to fire at the critical distance of 5000 meters, the distance at which it was designed to perform, the ability of the controller to hit its target increases dramatically. Figure 5.20 shows the results of these experiments:

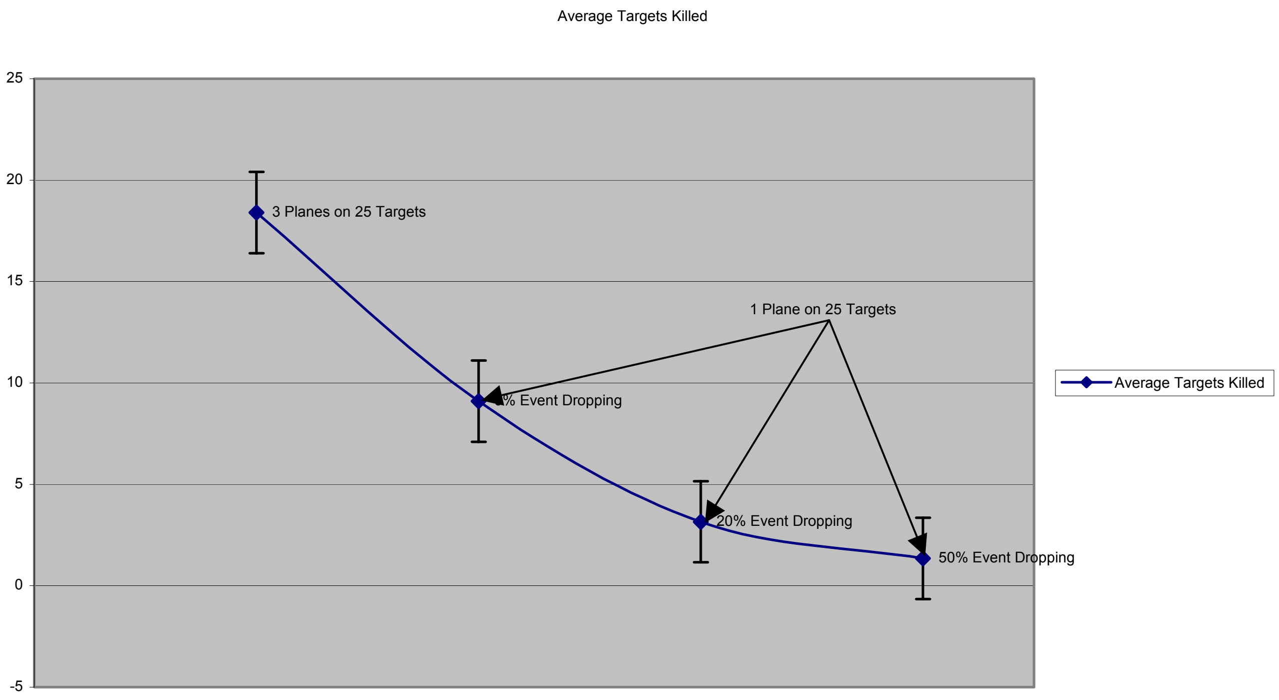
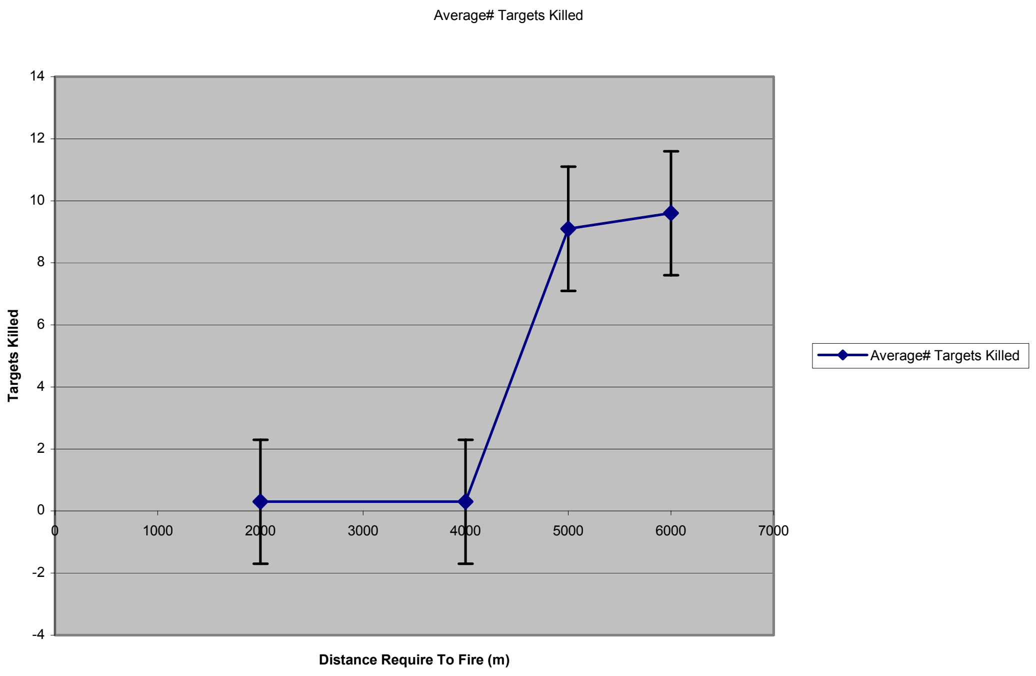


Figure 5.28: Results of Initial Experiments

This experiment is a starting point for further research into adaptive tactical intelligence. Assuming a commander (controller) could see and recognize the negative feedback produced if pilots cannot fire at 5000 meters (because of weather for example) it [the controller] might reconfigure its mission specifications to alleviate this problem or at the very least to find a work around.



Figur e 5.29: Sensitivity of Wild Weasel Performance to Ordnance Range.

5.6.3. Early Large Scale SEAD Experiments:

The results in this section have been produced by running Monte Carlo simulations of large scale SEAD scenarios. These results were the product of nearly 14 days of continuous experimentation. All of the results were guaranteed to have an error measure of less than or equal to 5%; of the four sets of experiments.

Figure 5.21 shows a graph of initial large-scale mission and their corresponding results. Here we are comparing four algorithms. One uses a tactical intelligence module to reconfigure when planes are destroyed. The other does not. We are also examining the effect lack of information has on overall mission success. In this case, planes were allowed to know 20% of the targets in their corridor before going out on missions.

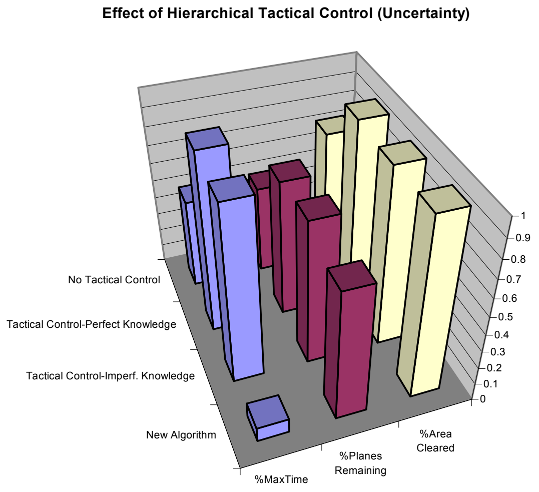


Figure 5.30: Early Large Scale SEAD Experiments

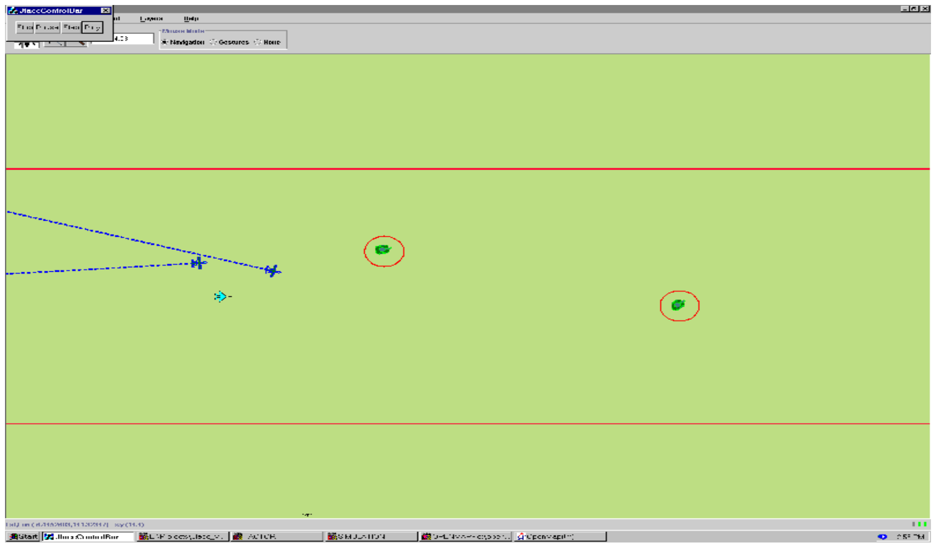


Figure 5.31: Current Visualization Interface

Other accomplishments include cosmetic changes to the C4I Simulator's Openmap™ interface and code for moving targets. Targets are capable of moving on their own in various patterns defined as methods and called from their "update" methods. Figure 5.31 shows moving targets and also demonstrates some cosmetic changes made to the SAM images.

Appendix A: Objective Function

Our experiment is a simulation of a limited SEAD scenario. A bombing mission is to be attempted against an enemy airbase. For the bombers to be able to perform the mission, enemy air defenses must be disabled in multiple corridors leading to the base. We will simulate the dynamics in establishing the corridors and keeping threats to friendly aircraft from the corridors.

For Preliminary Experimentation, we are limiting the number of entities involved. The enemy has three types of entities: (1) fixed SAM sites, (2) mobile SAM launchers, and (3) fixed radar sites. In the initial experiment, mobile SAM launchers perform a random walk of the terrain. They stop wandering and prepare to attack at random points in the walk. This logic may be augmented later. Any target that has been hit is disabled for a period of time following an exponential distribution with mean lambda, which will be fixed in the initial experiment.

Friendly forces are limited to Wild Weasels, which search for and destroy SAMs. Each wild weasel has its own on-board discrete event controller. The local controller has access only to local information. The higher-level controller interacts with the system by receiving ISR inputs and sending radio messages to the aircraft. At first a flat terrain is used and weather effects are ignored in the first iteration. For later experiments (after March) more complexity will be introduced.

In the scenario the corridors are given, they are four miles wide at their narrowest, to insure the safety of aircraft flying down the middle of the corridor.

Each aircraft starts with an initial mission to be completed. The aircraft's controller determines the decisions that are taken as events occur. Missions will be to patrol parts of the corridor and destroy enemy entities, adjusting the regions covered by each aircraft in response to changing conditions.

The objective function has four main fields:

1. Area of airspace (in cubic meters) kept free for a time unit (minutes)
2. Number and type of enemy targets neutralized
3. Number of friendly resources destroyed in the operation
4. Time required to clear the corridor

All these fields are relevant and need to be considered in the objective function. The objective function must therefore be a weighted non-linear function that simultaneously:

- Maximizes the expected value of the random variable $S = s(X(1), X(2), X(3)..., X(N), G)$ expressing the area and duration of portions of the corridor kept free from enemy threat.
- Minimizes the expected value of random variable $T = t(X(1), X(2), X(3)..., X(N), G)$ expressing the time needed to establish the corridor.
- Maximizes the expected value of random variable $U = u(X(1), X(2), X(3)..., X(N), G)$ giving the value of net enemy targets destroyed.
- Minimizes the expected value of random variable $V = v(X(1), X(2), X(3)..., X(N), G)$ expressing the risk (damage) taken by (inflicted on) friendly forces.

where $X(i)$ is the trajectory of friendly aircraft *i*. $X(i)$ is in terms of aircraft *i*'s position and velocity at all times. *G* is a known set of parameters describing actions and states of friendly and enemy forces.

When enemy targets (stationary or moving) are in the neighborhood of *i,* an event occurs that forces a decision to be made. The objective function is expressed quantitatively as:

$$C_1 f_1(E(S)) + C_2 f_2(E(T)) + C_3 f_3(E(U)) + C_4 f_4(E(V))$$

where E() denotes expected value of the random variable. C_j are time varying weights (costs) expressing the relative importance of factors *S, T, U,* and *V* as defined above. The weights sum to one. f_j are non-linear positive (negative) valued functions of the factors we wish to minimize (maximize) *T, V* (*S,U*).

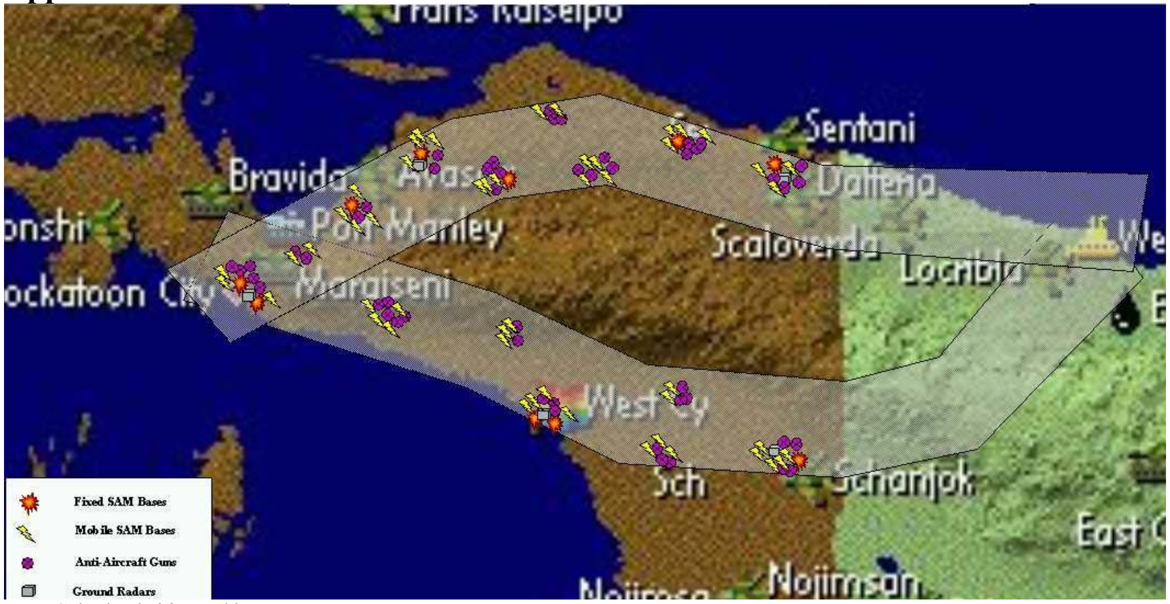
Other factors, such as weapon and fuel consumption could be added to this approach. One factor that we consider, which is not contained in this function, is the time required to make a decision. We are studying the use of resource-bounded optimization techniques for decision making. These techniques try to make the best decision given time and resource constraints.

In addition to possible improvements already given, later possible enhancements to the scenario include, but are not limited to:

- Addition of fuel constraints and tankers to re-fuel wild weasels.
- Addition of AWACS nodes for coordination of portions of the battlespace and the coordination necessary among the AWACS.
- Addition of fighter aircraft on both sides.
- Addition of bombers to destroy the airfield and escort fighters.
- Addition of jammers.
- Consideration of hierarchies of more levels of command.
- Evaluation of the influence of a full range of sources of error.

Based on the results of the first experiment, we will determine which of these should be added to the model.

Appendix B: Mission Overview



West Cyberland cities and bases

Rockatoon City	Location: 04:05S/134:43E"
Nahertonshi	Location: 03:26S/133:24E"
Port Manley	Location: 03:30S/135:12E"
Margiseni	Location: 03:47S/135:32E"
Nojimsan	Location: 07:13S/140:05E
Schanjok	Location: 06:01S/140:42E
Sentani	Location: 02:34S/140:31E
Dalteria	Location: 03:08S/140:36E
Avascir	Location: 02:54S/136:07E
Bravida	Location: 03:04S/134:26E

East Cyberland cities and bases

Wewak	Location: 03:35S/143:40E
Mount Hegan	Location: 05:50S/144:18E
Tuljeti	Location: 07:27S/141:54E
Locribla	Location: 03:52S/143:09E

Fixed SAM Bases

	Latitude	Longitude	Location
1.	"04:05S	134:42E"	Rockatoon City
2.	"04:15S	134:53E"	
3.	"03:05S	135:42E"	Between Port Manley and Avascir
4.	"02:60S	136:12E"	Avascir
5.	"02:54S	136:91E"	Group 1 East of Aviscar
6.	"02:34S	139:31E"	Group 1 West of Sentini
7.	"02:79S	140:31E"	Between Sentini and Dalteria
8.	"06:01S	140:40E"	Schanjok
9.	"05:70S	137:76E"	Group 3 W of Schanjok

Ground Radars

	Latitude	Longitude	Location
1.	"04:06S	134:41E"	Rockatoon City
2.	"02:61S	136:12E"	Avascir
3.	"02:78S	140:32E"	Between Sentini and Dalteria
4.	"06:01S	140:42E"	Schanjok
5.	"05:72S	137:75E"	Group 3 W of Schanjok

Mobile SAM Bases

	Latitude	Longitude	Location
1.	"04:10S	134:29E"	Rockatoon City
2.	"03:99S	134:31E"	

3.	"04:13S 135:15E"	
4.	"03:31S 135:18E"	Port Manley
5.	"03:41S 135:23E"	
6.	"02:99S 135:31E"	Between Port Manley and Avascir
7.	"03:01S 135:28E"	
8.	"02:94S 135:50E"	
9.	"03:03S 135:51E"	
10.	"02:53S 136:21E"	Avascir
11.	"02:62S 136:28E"	
12.	"02:51S 136:18E"	
13.	"02:49S 136:24E"	
14.	"02:53S 136:90E"	Group 1 East of Avascir
15.	"02:55S 136:88E"	
16.	"02:53S 136:92E"	
17.	"02:00S 137:77E"	Group 2 East of Avascir
18.	"02:01S 137:79E"	
19.	"02:64S 138:21E"	Group 3 East of Avascir
20.	"02:62S 138:20E"	
21.	"02:60S 138:19E"	
22.	"02:59S 138:20E"	
23.	"02:35S 139:29E"	Group 1 West of Sentini
24.	"02:36S 139:29E"	
25.	"02:34S 129:28E"	
26.	"02:35S 129:28E"	
27.	"02:80S 140:32E"	Between Sentini and Dalteria
28.	"02:81S 140:33E"	
29.	"02:78S 140:33E"	
30.	"05:99S 140:42E"	Schanjok
31.	"06:02S 140:40E"	
32.	"05:98S 140:41E"	
32.	"05:98S 140:42E"	
33.	"05:97S 140:39E"	
34.	"05:51S 140:31E"	Group 1 NW of Schanjok
35.	"05:50S 140:32E"	
36.	"05:80S 140:25E"	Group 2 W of Schanjok
37.	"05:81S 140:23E"	
37.	"05:79S 140:23E"	
38.	"05:70S 137:75E"	Group 3 W of Schanjok
39.	"05:69S 137:76E"	
40.	"05:68S 137:77E"	
41.	"05:71S 137:74E"	
42.	"03:55S 136:90E"	Group 2 E of Margiseni
43.	"03:56S 136:90E"	
44.	"03:54S 136:89E"	
45.	"03:45S 136:05E"	Group 1 E of Margiseni
46.	"03:44S 136:07E"	
47.	"03:47S 136:03E"	
48.	"03:50S 136:04E"	
49.	"03:51S 136:04E"	
50.	"03:52S 136:08E"	

Anti-Aircraft Guns

	Latitude Longitude	Location
1.	"04:07S 134:33E"	Rockatoon City
2.	"04:11S 134:58E"	
3.	"03:74S 134:51E"	
4.	"04:13S 135:13E"	
5.	"03:89S 135:14E"	
6.	"03:41S 135:22E"	Port Manley
7.	"03:36S 135:10E"	
8.	"03:02S 135:38E"	Between Port Manley and Avascir
9.	"03:09S 135:55E"	
10.	"02:80S 136:20E"	Avascir
11.	"02:71S 136:23E"	

12.	"02:52S 136:89E"	Group 1 East of Avascir
13.	"02:55S 136:87E"	
14.	"02:56S 136:92E"	
15.	"02:56S 136:91E"	
16.	"02:02S 137:76E"	Group 2 East of Avascir
17.	"01:99S 137:80E"	
18.	"02:00S 137:81E"	
19.	"02:61S 138:22E"	Group 3 East of Avascir
20.	"02:62S 138:20E"	
21.	"02:59S 138:19E"	
22.	"02:60S 138:22E"	
23.	"02:36S 129:27E"	Group 1 West of Sentini
24.	"02:33S 129:28E"	
25.	"02:34S 129:28E"	
26.	"02:36S 129:23E"	
27.	"02:79S 140:34E"	Between Sentini and Dalteria
29.	"02:80S 140:32E"	
30.	"02:81S 140:29E"	
31.	"02:82S 140:30E"	
32.	"05:98S 140:39E"	Schanjok
33.	"05:97S 140:41E"	
34.	"06:03S 140:42E"	
35.	"06:00S 140:42E"	
36.	"05:49S 140:31E"	Group 1 SW of Schanjok
37.	"05:52S 140:29E"	
38.	"05:53S 140:28E"	
39.	"05:81S 140:26E"	Group 2 W of Schanjok
40.	"05:82S 140:27E"	
41.	"05:83S 140:24E"	
42.	"05:71S 137:75E"	Group 3 W of Schanjok
43.	"05:69S 137:73E"	
44.	"05:70S 137:73E"	
45.	"03:55S 136:89E"	Group 2 E of Margiseni
46.	"03:53S 136:91E"	
47.	"03:45S 136:05E"	Group 1 E of Margiseni
48.	"03:47S 136:03E"	
49.	"03:41S 136:01E"	
50.	"03:39S 136:07E"	

Appendix C: Current Controller specifications

This section contains documentation on the controllers used in each experiment. It includes plant models, state and event definitions, low to high level language mappings, controller specifications, and controllers.

Experiment C1 – Controller for a single wild weasel

Level 1 – Design Controller for a Single Plane

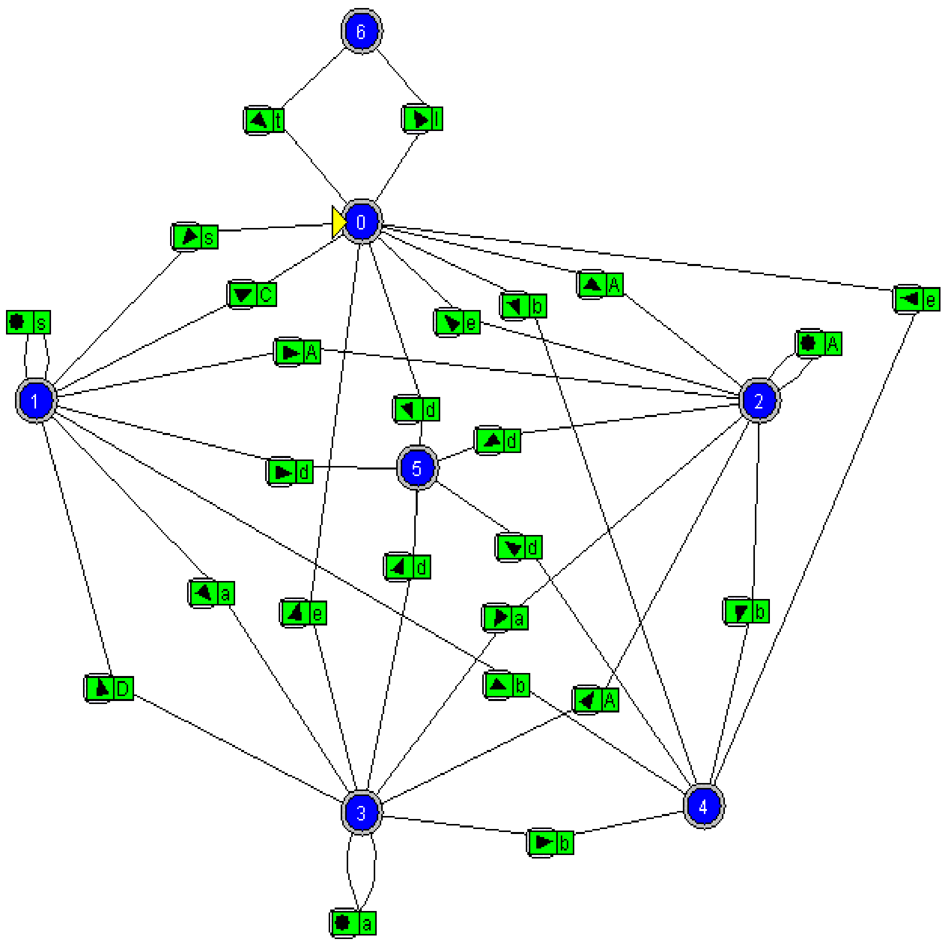


Figure C.1: Transition Diagram of FSM for Wild Weasel Plant Model

Events	Physical Meaning	Controllable
a	attack the target	Yes
A	alarm, enemy attack unsuccessfully	No
b	partial damaged	No
C	mission completed	No
d	the aircraft is destroyed	No
D	destroy the target	No
e	escape	Yes
l	all targets destroyed/go back for repair(mission abort)	Yes
s	search enemy	Yes
t	take off for a new mission	Yes
States	**Physical Meaning**	
q_0	Idle in air and safe (ready for mission)	
q_1	Searching for target	
q_2	Alarming that the fighter is being attacked	
q_3	Firing the missiles	
q_4	Damaged, can fly but cannot fight	
q_5	Get destroyed completely	
q_6	Mission completed/abort, back at base	

Table C-1. States and Events for Wild Weasel Plant Model

Wild Weasel Control Specifications
1. Try to fulfill the mission (*i.e.,* [*engage the enemy targets*]) unless more specific instructions are given from the requirements below.
2. At the initial state, q_0, start to [*search for the next enemy target*] if the mission is not completed.
3. If the aircraft is in the [alarming] state and 4 consecutive [*alarms*] have been received, [*escape*].
4. If two consecutive [*alarms*] have been received, [*fire*].
5. If the aircraft is in the [attacking] state and the target has not been destroyed after executing [*fire*] four consecutive times, [*escape*].
6. If the aircraft becomes [*damaged*], it should [*escape*].

Table C.2: Controller Requirements for a Single *Wild Weasel*

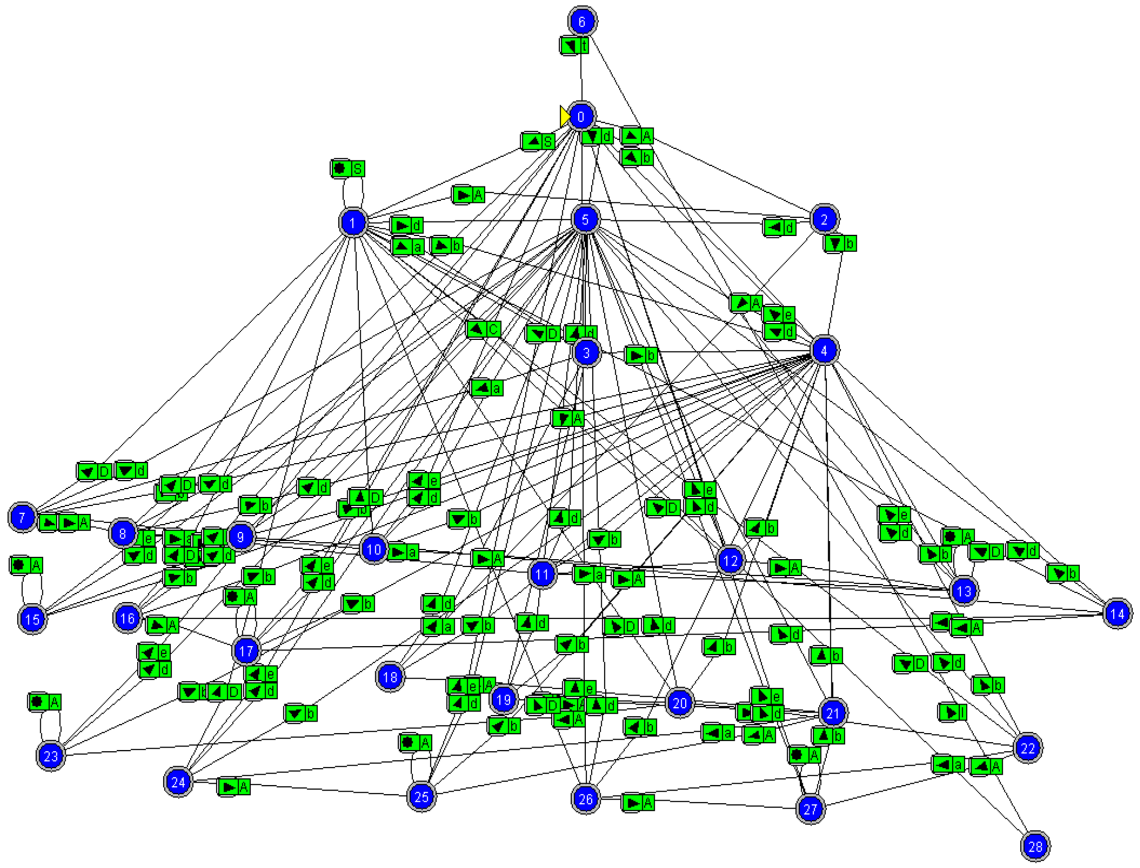
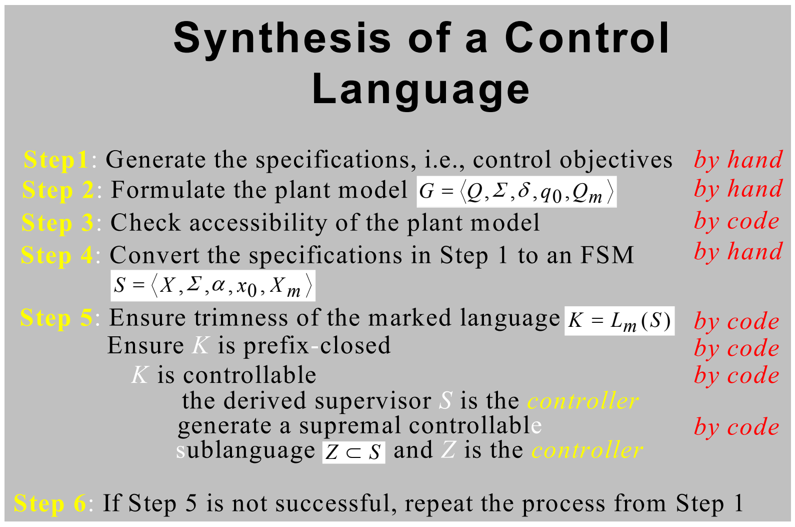


Figure C.2: Controller for a Single *Wild Weasel*

Level 2 – Verify Controller for a Single Plane



Synthesis of a Control Language

Step1: Generate the specifications, i.e., control objectives *by hand*

Step 2: Formulate the plant model $G = \langle Q, \Sigma, \delta, q_0, Q_m \rangle$ *by hand*

Step 3: Check accessibility of the plant model *by code*

Step 4: Convert the specifications in Step 1 to an FSM $S = \langle X, \Sigma, \alpha, x_0, X_m \rangle$ *by hand*

Step 5: Ensure trimness of the marked language $K = L_m(S)$ *by code*

Ensure K is prefix-closed *by code*

K is controllable *by code*

the derived supervisor S is the *controller*

generate a supremal controllable sublanguage $Z \subset S$ and Z is the *controller* *by code*

Step 6: If Step 5 is not successful, repeat the process from Step 1

Figure C.3: Steps for Testing Controllability

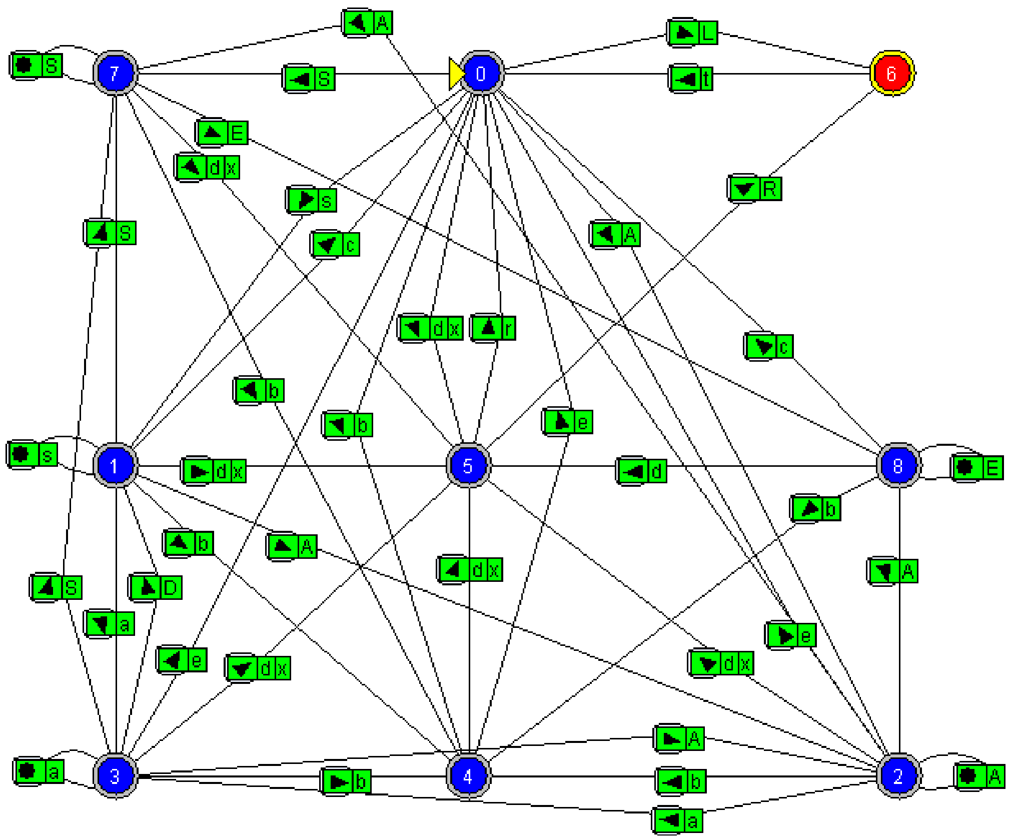


Figure C.4: Lower Level Plant Model for Hierarchical Control

Events	Physical Meaning	States	Physical Meaning
a	attack the target, i.e., fire	q_0	Idle in air and safe (ready for mission)
A	alarm, in the range of the target	q_1	Searching for target
b	partial damaged	q_2	Alarming that the aircraft is in danger
C	mission completed	q_3	Firing the missile
d	destroyed	q_4	Damaged, can fly but cannot fight
D	destroy the target	q_5	Get destroyed completely/unavailable
e	escape	q_6	Mission completed/abort, back at base
E	escort the partially damaged plane	q_7	searching for partially damaged friend plane

L	all targets destroyed/mission abort	q_8	Escorting friend plane
r	replace the destroyed plane		
R	reassign a new plane from the base		
s	search target		
S	search friend		
t	take off from the base		
x	transfer the plane to other group		

Table C.3: Lower Level Events and States for Hierarchical Control

Lower Level Control Specifications (for the operation of individual aircraft):	
1.	Try to fulfill the mission(i.e., [engage the enemy targets]) if none of the following situations occurred;
2.	At the initial state q0, start to [search for the next enemy target] if the mission is not completed except that the upper level controller asks for [searching for damaged friend aircraft] or [transferring to some other group]. In other words, the lower level controller should keep these three controllable transitions available: [search for the enemy target], [search for damaged friend aircraft], and [transfer to some other group]. It is upper level controller's responsibility to tell the lower level controller when it should enable one among these three commands;
3.	[Escape] if consecutive [alarms] from the enemy exceed 4 times since the first [alarm] was received;
4.	[attack/fire] after two or more consecutive [alarm] signals;
5.	[Escape] if consecutive [attack/fire] to the target exceed 4 times since the first [attack/fire] was taken;
6.	The [(partially) damaged] aircraft should stop mission and [escape] if capable to do so, the upper level controller will assign friend aircraft to [escort] this damaged one;
7.	[Search for damaged friend aircraft] and [escort] to base if the upper level controller asks to do so;
8.	During [searching for damaged friend aircraft], ignore [alarm] signals;
9.	During [escorting], if [alarm] comes in, then [attack/fire] the enemy only once and immediately come back and keep [escorting];
10.	[transfer to some other group] if the upper level controller asks to do so;
11.	Keep [replace current unavailable(either transferred or destroyed) aircraft by some one from other group] and [reassign a new aircraft from the base] available in the lower level controller. The upper level controller will decide when to enable one of these two commands.

Table C.4: Lower Level Control Specifications for Hierarchical Control

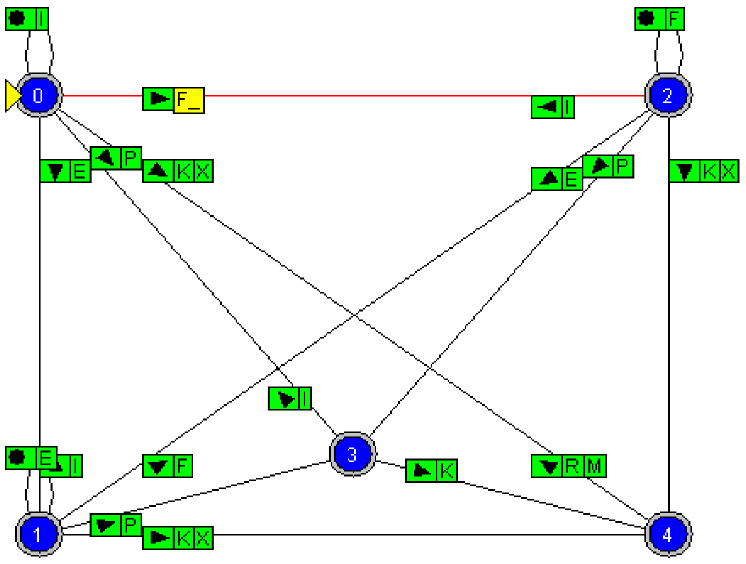


Figure C.5: Plant Model for Second Level of Hierarchical Control

Events	Physical Meaning	Controllable	Associated low event
E	engage	T	a, A, D, s
F	search friend and escort	T	S, E
I	disengage	T	c, e, L, t
K	plane gets killed	F	d
P	partially damaged	F	b
M	replace the plane	T	r
R	reassign a new plane	T	R
X	transfer to other group	T	x
States	Physical Meaning	States aggregation of lower level plant	
q'_0	disengaging	$\{q_0, q_6, q_9\}$	
q'_1	searching for friend and escort	$\{q_7, q_8, q_{32}\}$	
q'_2	engaging	$\{q_1, q_2, q_3, q_{10} \sim q_{31}\}$	
q'_3	partially damaged	$\{q_4\}$	
q'_4	destroyed/unavailable	$\{q_5\}$	

Table C.5: Events, States, and Mappings for Levels 1 and 2 of Hierarchical Control

1.	If a plane dies [destroyed] and the other is in [disengaging] state, replace the dead one with a new one before the other [engage];
2.	If both are [engaging] and then one dies [destroyed], the other [disengages] and not engage until new replacing one joins.
3.	If a plane is [partially damaged] it will be [escorted] back to base by its partner.
4.	Reduce the loss as far as possible when fulfilling the mission.

Table C.6: Second Level Control Specifications (for the operation of a group of aircraft)

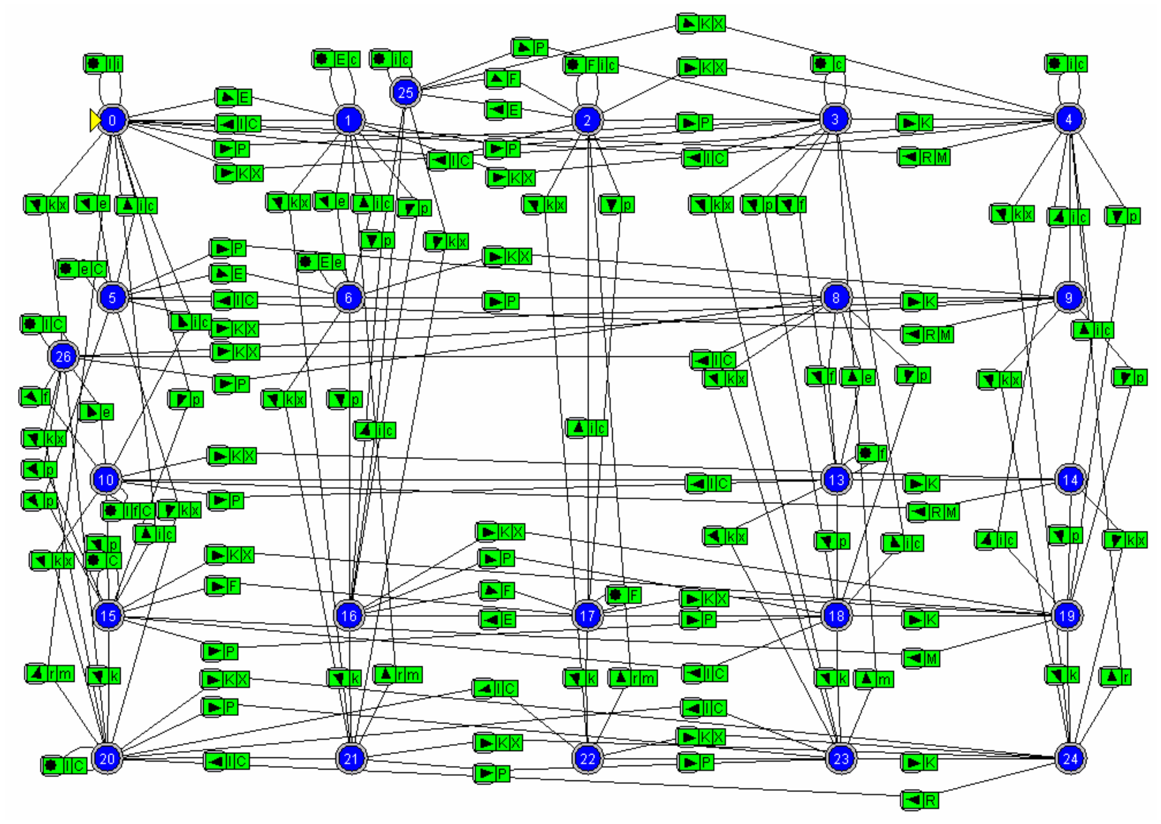


Figure C.6: Second Level Controller

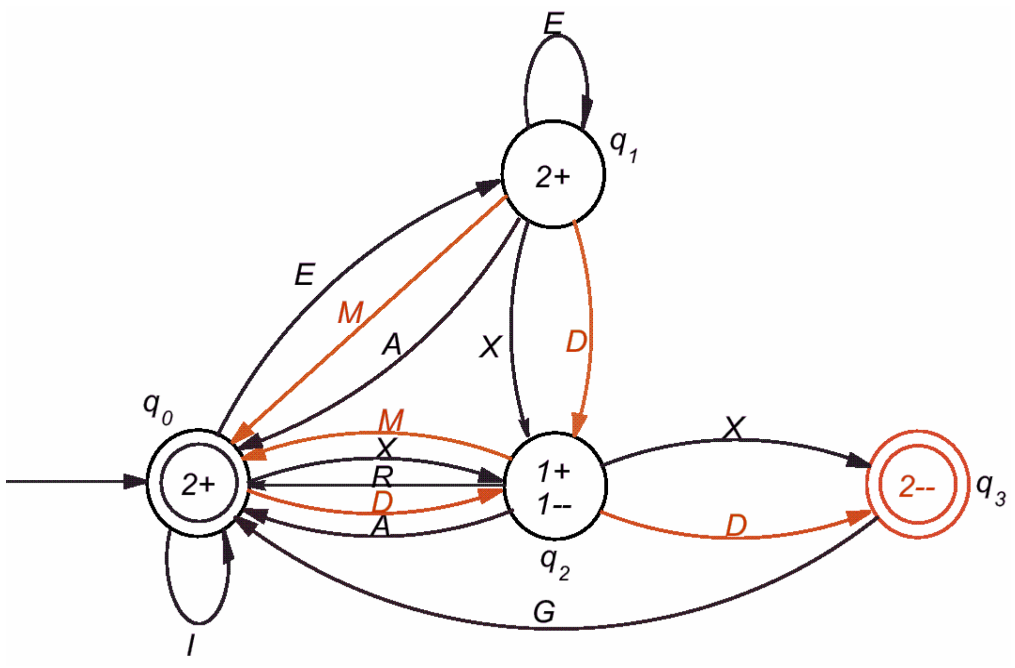


Figure C.7: Top Level Plant for 3 Level Hierarchy

List of Events: Controllable = {A, E, G, I, R, X} Uncontrollable = {D, M}
A: Mission Abort
D: Aircraft Destroyed
E: Engage
G: Replace Group
I: Idle
M: Mission Complete
R: Replace Dead Aircraft
X: Transfer Aircraft
List of States: Q = {q0, q1, q2, q3}
Q0: Idle (Both aircraft alive)
Q1: Engaging (Both aircraft alive)
Q2: One Alive, One Dead
Q3: Both Dead

Table C.7: States and Events for Top Level Controller

From State	Up/ Low Event	To State	From State	Up/ Low Event	To State
0	I/I or I/i	0	all states	D/K	4/9/14/19
0	E/E	1	all states	D/k	20/21/22/23
0	E/e	5	all states	X/X	4 or 9 or 14 or 19
19	D/k	24	all states	X/x	20 or 21 or 22 or 23
23	D/K	24	24	G/R	20
4/9/14/19	R/R	0/5/10/15	24	G/r	4
20/21/22/23	R/r	0/1/2/3	all states	M/C or M/c	

Table C.8: Mealy Machine Construction for Middle-Level Individual Group of Two Aircraft

1.	Try to fulfill the mission;
2.	Allow mission abort (When? tactical intelligence?);
3.	If both groups have one dead and one alive, then form a new group;
4.	If one group is dead, replace the group;
5.	If only one aircraft is dead in a group, replace the dead aircraft with a new one that comes from either the base or another group.

Table C.9: Third Level Specifications

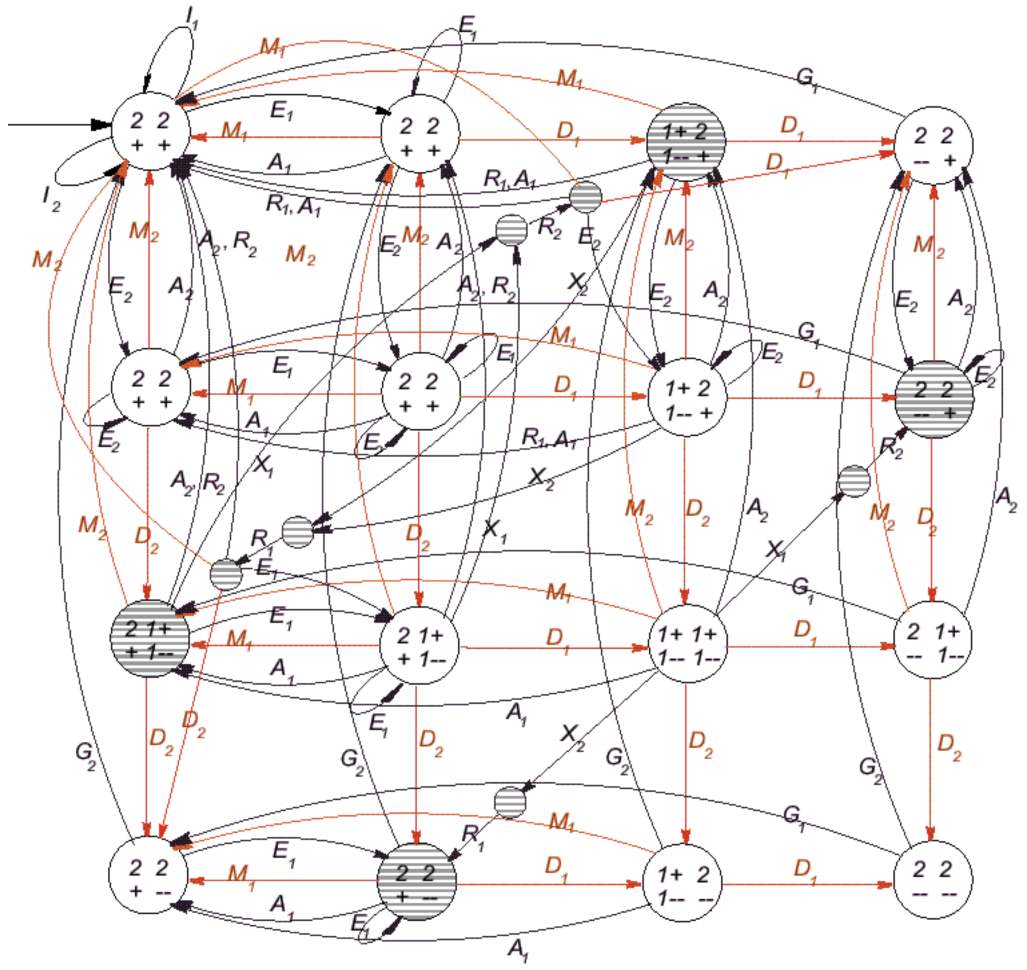


Figure C.8: Third Level Controller Diagram